DEGREE PROJECT IN MATHEMATICS,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2018

Recurrent neural networks for financial asset forecasting

GUSTAF TEGNÉR

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ENGINEERING SCIENCES

Recurrent neural networks for financial asset forecasting

GUSTAF TEGNÉR

Degree Projects in Mathematical Statistics (30 ECTS credits)
Degree Programme in Applied and Computational Mathematics (120 credits)
KTH Royal Institute of Technology year 2018
Supervisor at Lynx Asset Management: Martin Rehn
Supervisor at KTH: Timo Koski
Examiner at KTH: Timo Koski

TRITA-SCI-GRU 2018:262
MAT-E 2018:60

Royal Institute of Technology
School of Engineering Sciences
KTH SCI
SE-100 44 Stockholm, Sweden
URL: www.kth.se/sci

Sammanfattning

Tillämpningen av neurala nätverk i finans har fått förnyat intresse under de senaste åren. Neurala nätverk har en erkänd förmåga att kunna modellera icke-linjära förhållanden och har bevisligen visat sig användbara inom områden som bild och taligenkänning. Dessa egenskaper gör neurala nätverk till ett attraktivt val av model för att studera finansmarknaden

Denna uppsats studerar användandet av rekurrenta neurala nätverk för prediktering av framtida prisrörelser av ett antal futures kontrakt. För att underlätta får analys jämför vi dessa nätverk med en uppsättning av enkla framåtkopplade nätverk. Vi dyker sedan djupare in i vår analys genom att jämföra olika målfunktioner för nätverken och hur de påverkar våra nätverks prestation. Vi utökar sedan den här diskussionen genom att också undersöka multi-förlust nätverk. Användandet av flera förlust funktioner visar på betydelsen av vårt urval av attribut från indatan. Vi studerar ett par simpla och komplexa attribut och hur de påverkar vår modell. Det hjäper oss att göra en ytterliggare jämförelse mellan våra nätverk. Avslutningsvis så undersöker vi vår modells gradienter för att få en utökad förståelse över hur vår modell agerar med olika attribut.

Resultaten visar på att rekurrenta nätverk utpresterar framåtkopplade nätverk, både i uppgiften att maximera sharpe ration och precision. De enkla attributen visar på bättre resultat när nätverket optimeras för precision. När vi optimerar för att maximera Sharpe ration fungerar de komplexa attributen bättre. Tillämpningen av multi-förlust nätverk visade sig framgångsrik när vårt huvudmål var at maximera sharpe ration. Våra resultat visar på en signifikant ökad prestation av våra nätverk jämfört med ett par enkla benchmarks. Genom ensemble metoder uppnår vi en Sharpe ratio på 1.44 samt en precision på 52.77% på test datan.

Abstract

The application of neural networks in finance has found renewed interest in the past few years. Neural networks have a proven capability of modeling non-linear relationships and have been proven widely successful in domains such as image and speech recognition. These favorable properties of the Neural Network make them an alluring choice of model when studying the financial markets.

This thesis is concerned with investigating the use of recurrent neural networks for predicting future financial asset price movements on a set of futures contracts. To aid our research, we compare them to a set of simple feed-forward networks. We conduct further research into the various networks by considering different objective loss functions and how they affect our networks performance. This discussion is extended by considering multi-loss networks as well. The use of different loss functions sheds light on the importance of feature selection. We study a set of simple and complex features and how they affect our model. This aids us in further examining the difference between our networks. Lastly, we analyze of the gradients of our model to provide additional insight into the properties of our features.

Our results show that recurrent networks provide superior predictive performance compared to feed-forward networks both when evaluating the Sharpe ratio and accuracy. The simple features show better results when optimizing for accuracy. When the network aims to maximize Sharpe, the complex features are preferred. The use of multi-loss networks proved successful when we consider achieving a high Sharpe ratio as our main objective. Our results show significant improved performance compared to a set of simple benchmarks. Through ensemble methods, we achieve a Sharpe ratio of 1.44 and an accuracy of 52.77% on the test set.

Acknowledgements

I would like to thank my supervisors Martin Rehn, Max Nyström Winsa and Jing Fu at Lynx Asset Management for providing invaluable insight into the intuition behind neural networks and for the continuous feedback and comments I received throughout my time there.
I would also like to extend my gratitude to Timo Koski at KTH Royal Institute of Technology for supervising this project.

Stockholm, June 2018

Gustaf Tegnér

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Neural networks have gained immense popularity over the past few years for their ability to allow modeling without a preconception of an underlying structure in the data. The financial market is a domain that is notoriously hard to model using traditional methods. Due to neural networks unsupervised approach to modeling data, it is possible that we could find non-linear relations in the markets which past methods fail to find. A promising type of network is the Recurrent Neural Network which in contrast to simple feed-forward networks, is able to take into account the time dependency inherent in financial data.

We aim to analyze both recurrent and feed-forward networks on a set of futures contracts on commodities, rates and FX contracts. Equipped with this proposition, we are immediately faced with two issues. If we want to understand futures trading and the financial market, a statistical model that is interpretable is preferred compared to a black-box model such as a neural network. If we want to investigate various neural networks, highly noisy data coming from financial sources make it hard to perform informative experiments. By considering different feature setups and model architectures, this thesis aims to walk the fine line between these two contradictory statements and provide a detailed analysis of the relation between neural networks and financial markets.

1.1 Research questions

Throughout this thesis we seek to gain a deeper understanding into the nature of the market and how to explicitly tune a set of neural networks to exploit possible inefficiencies. In particular we will limit our research to the following set of questions.

1. What set of features are most significant to the task of predicting future returns?
2. Can a recurrent neural network outperform a Feedforward network given the same feature setup?
3. What is the influence of the objective function of the Neural Network and how does it impact feature selection?

1.2 Related work

This thesis stands on the shoulders of Philip Widegren's Masters Thesis "Deep Learning-based forecasting of Financial Assets" [18]. Widegren investigates and compares the use of Feedforward-Networks and simple Recurrent Networks for application of financial asset forecasting. Our thesis serves as a direct extension of Widegrens work hence a great deal of the methodology is inspired by the paper to enable us to provide a comparison between our different methods.

Further work has been done in the intersection between machine learning and finance by various authors. Another masters thesis that touches upon this subject is Magnus Hansson's Thesis "On Stock Market Prediction using LSTM Networks" [8]. Hansson compares the use of an advanced type of recurrent network, the Long-Short Term Memory (LSTM) network, to more traditional auto-regressive moving average models. The thesis investigates both the models performance across different markets with respect to directional accuracy as well as evaluating trading strategies based upon the accuracy of its predictions. The author concludes that the trading strategies generated by the LSTM outperform the conventional time-series models.

An interesting paper that concerns time-series prediction is "Deep and Confident Prediction for Time Series at Uber" [19]. The authors predict the number of trips given different weekdays and holidays together with developing a framework for anomaly detection using LSTM networks. The most interesting aspect of the paper is the emphasis of utilizing LSTMs for automatic feature extraction in time series data.

Another paper which attempts to apply Deep Learning to Finance is "A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem" by Z. Jiang et al [12]. Deep Reinforcement Learning has found popularity in tasks such as learning to play video games directly from the pixel data. In this paper the authors apply it to create automatic trading bots from intraday data. To learn a video game with only the pixels as input, an RL agent uses neural networks created for image recognition to determine the players *state* on the board. Similarly, in the case of financial markets, the authors uses among other methods, LSTMs, directly on financial data to determine the trading bots state in the market. One of the conclusions the authors draw is that LSTMs perform worse than other networks and specifically that simple Recurrent Networks and Convolutional Neural Networks outperform them.

1.3 Scope and Limitations

The thesis focuses on understanding and predicting financial markets through the use of neural networks. One limitation of our research is hyperparameter optimization. We have searched through a relatively large space of hyperparameters together with training each network with different random seeds to provide a measure of stability as well. This is a process that can always be explored further to provide even more accurate results.

1.4 Outline

We employ the following structure to our thesis. In chapter 2 we begin by giving the necessary background information to understand our data and an idea of why financial markets are hard to predict. Chapter 3 dives into the construction and optimization of feed-forward and recurrent neural networks. Chapter 4 gives a detailed explanation of our methodology including feature extraction and preprocessing for our data as well as an insight into what performance metrics we consider for our networks. We also mention what architectures we wish to explore and how we evaluate the importance of a feature in our model. We present our results together with a detailed analysis of them in chapter 5. The discussion in chapter 6 then begins with a more general comment on our methodology and results where discuss the effectiveness of our method. Chapter 6 then proceeds with a discussion on how well we have researched our initial research questions and again pinpoint the limitations of our analysis. We end the discussion chapter with a note on possible future work that could be used to extend this thesis. A final conclusion is presented in chapter 7.

Chapter 2

Financial Background

Financial data is some of the most difficult data to model. In this chapter we give the necessary background to understand the data in this thesis as well as outlining the difficulties that arise when trying to model financial data.

2.1 Futures

For our input data, we will consider a special type of financial contract called Futures contracts. A section on future contracts can not be complete without and introduction to the forward contract. Consider an underlying asset X which could for example be a commodity such as gold. A forward contract on X, made at time t with a delivery date at T is a contract which stipulates the holder to receive a fixed amount of the underlying asset at time T for the **forwardprice** K. The forward contract thus ensures that the holder is mitigated from the risk of price movements in the underlying asset. In the forward contract the transaction is settled at the delivery date, when the forward price K is paid to receive the underlying. This brings us to the futures contract. With the same setup as before, the futures contract differs from the forward by the fact that all payments from the holder of the contract to the underwriter are made continuously until the delivery date T is reached. Let $F(t,T,X)$ denote the *futures price* at time $t < T$. At time $t + \delta t$ the holder of the contract receives the difference

$$F(t + \delta t, T, X) - F(t, T, X). \tag{2.1}$$

As the contract is settled continuously, the value of price of the futures contract at any time is 0. Hence, it is free to enter or leave the contract at any time, the only obligation is the payment stream.

2.2 Efficient Market Hypothesis

This thesis aims to evaluate the performance of Recurrent Neural Networks on the basis of its ability to predict future market movements. The input to our model is the aggregated past price movements of several future contracts. The Efficient Market Hypothesis (EMH) concerns the general claim that given any collection of information about the market, is it at all possible to predict the future? The EMH is mainly associated with two claims:

1. Price changes in financial markets are random
2. The price today reflects all available information about the market.

This leads to the market being considered *efficient*. There are different forms of the EMH which concerns what to include as "all the available information". Specifically, this refers to three variants of efficiency: weak, semi-strong and strong.

2.2.1 Weak Efficiency

The weak form of the EMH states that a market exhibits weak-form efficiency if the current price of the security completely includes all historic public information about historic prices, trading volume and other information available from the market only. A consequence of this is that future behaviour is uncorrelated with the past. Another name for the weak form efficiency is unsurprisingly the random walk theory.

2.2.2 Semi-strong form

The weak-form efficiency considers public information limited to historic order book data. The semi-strong efficiency considers all publicly available information. That is, in addition to the claims of the weak-form, it considers non-market information such as earnings and dividends information as well as both news data about the market, economy and politics. While the weak-form states that analyzing past prices is useless, the semi-strong form goes one step further and considers even *fundamental analysis* to be a futile endeavor.

2.2.3 Strong form

In addition to historic price data and non-market data the Strong form efficiency states that the current price even reflects all *insider information* that is not considered to be public information. There is little reason to believe that the market exhibits strong form efficiency, one reason being that insider trading continues to happen.

2.2.4 Adaptive Markets

Even though strong-form efficiency and even semi-strong efficiency seem improbable, there is compelling evidence that the weak-form holds. Still, the fact remains that some hedge-funds, trading firms and individual investors have been shown to consistently beat the market. The most notable example being the fully systematic fund Renaissance Technologies which has shown an annualized 35% return over a 20 year period [15]. This type of performance is rare, but it is something that is not covered by the EMH. The rising discipline of behavioural economics has sought to explain the market as being driven by a large part by fear and greed and are thus not at all times rational. A more nuanced theory that extends the EMH is the theory of *Adaptive Markets* [14]. The Adaptive Markets Hypothesis takes into consideration the irrational behaviour of investors and states that:

> Prices reflect as much information as dictated by the combination of environmental conditions and the number and nature of "species" in the economy.

"Species" refers to different market participants such as hedge funds, retail investors and pension funds. The theory states that the degree of market efficiency varies between different time periods, markets and the adaptability of market participants. From this, one can derive a number of consequences that puts the theory at odds with the EMH:

1. Arbitrage opportunities exist
2. Fundamental analysis, technical analysis and other quantitative strategies are able to perform well in certain environments.
3. The relationship between risk and reward, the price of risk, is constantly changing over time. An adaptive strategy can thus be deemed better to achieve a consistent level of expected returns.

Hence, according to the Adaptive Markets Theory we can possibly expect to see evidence of increased prediction ability in different time-periods, possibly indicating that the rate of market efficiency is lower in various market conditions.

2.3 Stationarity

The notion of market efficiency can be related to the mathematical notion of *stationarity*. Let X_t denote a time series and let $F_X(x_{t_1+h}, \ldots x_{t_k+h})$ represent the joint cumulative probability distribution of X_t at times $t_1 + h, \ldots, t_k + h$. X_t is then said to be *strictly* stationary if

$$F_X(x_{t_1+h}, \ldots x_{t_k+h}) = F_X(x_{t_1}, \ldots, x_{t_k}) \quad \forall k, h, t_1, \ldots, t_k \tag{2.2}$$

In other words, a time series X_t is strictly stationary if the distribution of the data is the same for all time periods.
There is a weaker form of stationarity as well, which states that for a time series is *weakly stationary* if

$$E[X_t] = \mu \tag{2.3}$$

$$Cov(X_t, X_{t+h}) = \gamma_h. \tag{2.4}$$

In other words a time series X_t is weakly stationary if its expected value is constant and the covariance with past data is only dependent on the lag-factor h and not which time period t is taken into consideration. Note that strong condition implies weak stationarity. An example of a non-stationary time series is the random walk

$$X_t = X_{t-1} + \varepsilon_t \tag{2.5}$$

$$\varepsilon_t \sim \mathcal{N}(\prime, \infty) \tag{2.6}$$

The weak form of the efficient market hypothesis states that markets exhibit behaviour similar to random walks. A stationary financial time series would thus contradict the EMH. We cannot expect a financial time series to display stationarity throughout the entire dataset.

Chapter 3

Neural networks

The name Neural-Net stems from its history of trying to find a mathematical representation of the information processes of the brain. The artificial neural network consists of interconnected groups of artificial neurons. The neurons fire through an activation function which represents the similar function in the brain where neurons either activate or not depending on which information they receive. Learning in neural networks is composed of updating the connections between the artificial neurons. This is similar to how the human brain updates the synaptic connections between the neurons. Our focus is not whether these are accurate representations of biological information processing systems but rather their strength as devices for pattern recognition.

Neural networks have gained popularity due to the fact that they are able to effectively model data using very limited assumptions on the underlying distribution and discover patterns that too complex for humans or other algorithms to find. In classical statistical modeling, the data aids in the process of model selection. We can note that the data seems to be linear or polynomial or exhibit other properties which allows us to make educated guesses on the underlying distribution and what model would be best to serve our purpose. The data is then used to estimate the parameters in this model to at last compare this fitted model to other alternatives. In the world of Neural Networks, the role of data is slightly different. Data is used to directly model the underlying function which generates the data. The clear advantage is that we need no assumption of the distribution of data since the neural networks itself discovers it.
Thus, another term for the neural net could be *universal function approximators*. It can be shown that a *feed-forward neural network* with 1 hidden layer and a finite number of neurons can approximate any continuous function on compact subsets of $\mathbb{R}^n$.

3.1 Feed-forward neural Networks

Mathematically, the neurons in an artificial neural networks is represented by the weight matrices W^k. The activations are *non-linear* functions f which usually squash the input into a smaller range. The number of *hidden-layers* is determined by how many matrix multiplications and non-linearities exist between your input-layer and output layer. We can describe the architecture of the feed-forward network succinctly by the following process of calculations.

Given an input $X \in \mathbb{R}^d$, the output y of a L layer neural net is found by an iterative process of matrix multiplications and non-linearities. This is called a forward pass of the network and is defined as
Definition 3.1. *Forward pass*

$$h_1 = f^1(W_{(1)}^T X + b_l) \tag{3.1}$$

$$h_l = f^l(W_{(l)}^T h_{(l-1)} + b_l) \tag{3.2}$$

$$y = f^{L+1}(W_{(L+1)}^T h_{(L)} + b_{L+1}) \tag{3.3}$$

The feed-forward neural network is thus a composition of functions $f^l \circ W^{(l)T}$. We can now arrive at the Universal Approximation Theorem
Theorem 3.2. *Universal Approximation theorem Let $\rho(\cdot)$ be a non-constant, bounded and monotonically-increasing continuous function. Let I_m be an m-dimensional unit hypercube $[0,1]^m$. The space of continuous functions on I_m is denoted $C(I_m)$. Then, given any $\varepsilon > 0$ and any function $f \in C(I_m)$, there exists an integer N, real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathcal{R}^m$, where $i = 1, \ldots, N$ such that we may define*

$$F(x) = \sum_{i=1}^{N} v_i \rho(w_i^T x + b_i) \tag{3.4}$$

as an approximate realization of the function f where f is independent of ρ; that is,

$$|F(x) - f(x)| < \varepsilon \tag{3.5}$$

for all $x \in I_m$. In other words, functions of the form $F(x)$ are dense in $C(I_m)$

Note that equation 3.4 is exactly the operation of a one-layer feed-forward neural network. We can thus conclude that a feed-forward neural network has the capacity of approximating almost any function.
This may seem like a gift, but it imposes some limitations on the benefit of using a neural network for statistical analysis. Given a task such as classification, each layer transforms the data in such a way that distinct properties of

each class can be extracted to further aid the classification that is performed in the last layer. The issue is that how exactly this is performed is still an open research question. Neural Networks are also considered *non-identifiable* models. Training two equivalent neural networks on the same task can lead to different values of the weights which further makes the model difficult to interpret. Neural Nets are for this reason called black-box models. Some applications of neural networks can range from predicting customers risk-premiums for a insurance company to identifying brain damage from MRI-scans. Common to both of these applications is that the model needs to have a level of interpretability to allow experts to motivate and verify the decision of the model, otherwise the models decision could possibly lead to life threatening consequences in the case of applications within health-care.

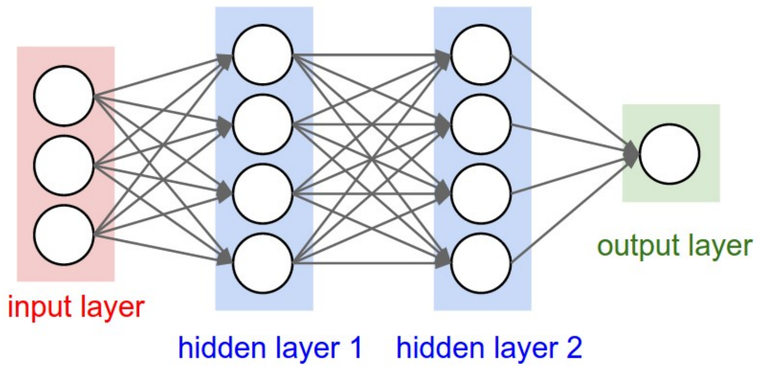


Figure 3.1: Architecture of a two layer feed-forward network. The circles represent neurons and the connections between them synapses.

3.2 Recurrent Neural Networks

Recurrent Neural Networks (RNN) can best be described as an extension of vanilla feed-forward networks with the property that the output of the model serves as an input together with the next training example. This property makes RNNs suitable for tasks which involve a time series such as speech recognition and machine translation. The idea of RNNs stem from the fact that humans refer to previous information when performing a task. When we read a sentence, we do not evaluate each individual word as they are but evaluate them given the previous words in the sentence. To model this sort of memory, recurrent neural networks keep a hidden state h_t which serves as a vector that summarizes the information in the previous inputs.

There are several different types of input-output architectures for RNNs which are chosen given the nature of our problem. There are *sequence input and non-sequence output* models which can for example be used to classify what genre a song belongs to. There are also *sequence input to sequence output* models which are prevalent in translation tasks. A third model would be non-sequence input to sequence output. These work by for example *generating* descriptions of a given image.

3.2.1 Simple Recurrent Neural Network

Consider a sequence of inputs $x_1, \dots, x_T$ with $x_i \in \mathbb{R}^D$. At time-step t, the RNN takes the current input x_t together with the previous *hidden state* $h_{t-1} \in \mathbb{R}^p$ to calculate the output. In particular, at each time-step we apply the recurrence formula

$$h_t = f_W(x_t, h_{t-1}) \tag{3.6}$$

where f_W is a function with parameters W. These parameters are the same for every time-step. The output can then be computed as

$$y_t = g_{W_2}(h_t) \tag{3.7}$$

where g_{W_2} can be modeled as a simple feed-forward net.

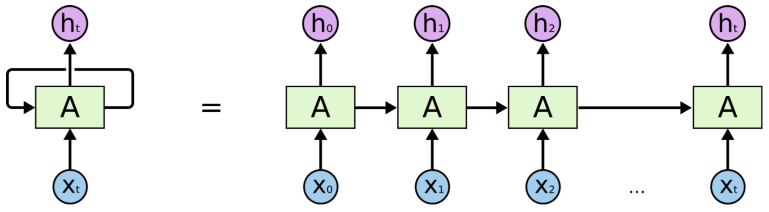


Figure 3.2: RNN

Simple RNNs are thus simply a sequence of regular feed-forward networks where the output is a hidden state.

Simple RNNs have shown to be effective in modelling non-linear time series. However, they exhibit a few issues which need to be addressed:

1. As the gradient is propagated through time, RNNs can have difficulties modeling long time-series because of vanishing or exploding gradients
2. Simple RNNs can have problems modeling longer term dependencies in the time-series.

3.2.2 Long Short-Term Memory

When reading a book, we not only keep track of individual words at a sentence level, we also remember characters and events mentioned several pages back and even the theme of the book to evaluate the precise meaning of what is happening at this moment. There are in other words multiple levels of memory stored in our thoughts which we at each time-point pick and choose the most relevant thoughts from to aid our task. To model this, and address the issues of the Simple RNN, Schmidhuber [11] developed the Long Short-Term Memory Recurrent Neural Network (LSTM). These are members of the class of RNNs which are called *gated RNNs*. Regular RNNs can accumulate information. The previous state is gradually accumulated throughout the time-series. Sometimes it can however be beneficial to *forget* some of this information. In the domain of analysis of financial time-series, where we deal with highly non-stationary data, we would ideally want to forget time periods where the distribution of the data differs from our current distribution. To achieve this, the LSTM works with a set of *gates* which regulate how previous information should be incorporated into the current output. In particular, the architecture is composed of a memory cell together with an input gate, output gate and forget gate. The gates produce an activation, usually created through a sigmoid function, which controls the extent of which new information flows into the cells. In particular we have:

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \tag{3.8}$$
$$i_t = \sigma_g(W_i x_i + U_i h_{t-1} + b_i) \tag{3.9}$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \tag{3.10}$$
$$c_t = f_t c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \tag{3.11}$$
$$h_t = o_t \circ \sigma_h(c_t) \tag{3.12}$$
$$\tag{3.13}$$

In the LSTM the hidden state h_t is created through the context vector c_t which passes through the *output gate* o_t. The context vector in turn consists of the previous context c_{t-1} together with the gated output of the previous input and hidden state. The context vector is thus a description of the history of the time series weighted with the new input. The forget and input gates f_t, i_t control how much previous versus new information should be included in the context. The output gate o_t then decides how much of this context should be taken into consideration when constructing the current hidden state h_t. This set of operations becomes more apparent in Figure 3.3

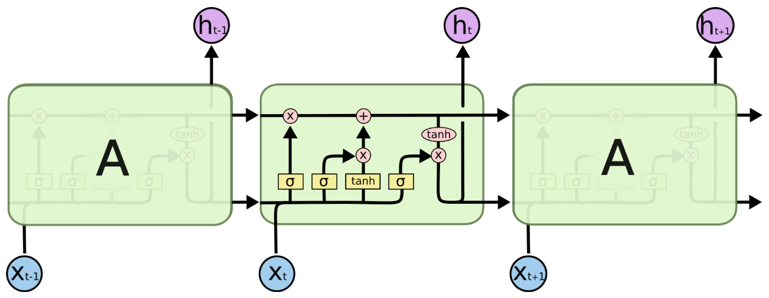


Figure 3.3: LSTM unit. Merging arrows describes concatenation. The upper flow is the flow of the context vector, the lower part the flow of h_t. Merging arrows as seen in the lower left part denote concatenation. The concatenation follows from the fact that $W_f x_t + U_f h_{t-1} = W_f^*[x_t, h_{t-1}]$ for matrix $W_f^* = [W_f, U_f]$.

From the figure, the function of the LSTM unit can be described as a series of pipes which control the flow of information. The context c_t, also referred to as the *internal memory* of the LSTM flows through the forget gate which determines how much of the memory we should keep. Next it passes through a summation sign which determines how much *new* memory we should include. The input x_t and hidden state h_{t-1} thus mainly serve to help us determine how much to alter the context vector.
An issue mentioned with simple RNNs is the problem of vanishing or exploding gradients. The magnitude of gradients is affected by the derivatives of the activation functions and weight matrices.

3.2.3 Gated Rectified Units

An alternative to Long-Short Term Memory units is the Gated Rectified Units (GRU) [2]. As the name suggests, these types of networks also utilize gates to process the input together with the previous output. Let h_t, x_t denote the hidden state and input respectively. The gates of the GRU is calculated as,

$$z_t = \sigma_g(W_z x_t + U_z h_t + b_z) \tag{3.14}$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{3.15}$$
$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_{h)} \tag{3.16}$$

where $\circ$ denotes the Hadamard product. The activation functions σ_g, σ_h were originally set to be the sigmoid function and the hyperbolic tangent respectively. There are a few advantages of using GRUs compared to LSTMs:

1. Fewer parameters due to a smaller number of gates. This leads to faster training.

2. Have been shown to exhibit better performance on smaller datasets. [4]

3.3 Residual Connections

For deep layers with many hidden layers it is common to encounter the problem of vanishing or exploding gradients. During training, gradients are propagated throughout the network by the chain rule. Thus, if the gradients are small, they will eventually vanish given a deep enough network. Residual Connections or skip connections was introduced by Microsoft Research [9] to prevent this and ease the training process. Adding residual connections between the layers can be expressed as

$$\hat{h}_l = F(W, h_{l-1}) \tag{3.17}$$

$$h_l = \hat{h}_l + h_{l-1} \tag{3.18}$$

$$\tag{3.19}$$

Residual Connections are motivated by the fact that if we want to learn an underlying mapping $\mathcal{H}(x)$, we can equivalently approximate the residual $F(x) = \mathcal{H}(x) - x$ instead. The original function thus becomes

$$\mathcal{H}(x) = F(x) + x. \tag{3.20}$$

In theory, deeper networks should identify redundant layers by letting them converge to identity functions during training. In practice, this is not guaranteed and by adding an identity function to the layer, the network can more easily identify if the layer is necessary or not.

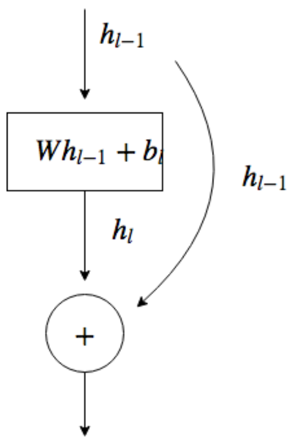


Figure 3.4: An overview of residual connections.

3.4 Loss functions

3.4.1 Binary cross-entropy

We are now faced with the task evaluating the performance of our network and finding a way to tune it's parameters to more accurately model its given task. Let us consider the problem of *binary classification*. Let $\hat{y}_i$ represent the probability of input x_i belonging to class 1 with y_i representing the true class of x_i. The conditional probability of the targets given our inputs could thus be modelled as a Bernoulli distribution with probability density function

$$P(y_i|X_i) = \hat{y}(w, x_i)^{y_i}(1 - \hat{y}(w, x_i))^{1-y_i} \tag{3.21}$$

The likelihood of our targets given our data is thus

$$L(y|X) = \prod P(y_i|X_i) = \prod \hat{y}(w, x_i)^{y_i}(1 - \hat{y}(w, x_i))^{1-y_i} \tag{3.22}$$

Considering the negative log-likelihood, we get

$$l(y|X) = -\sum y_i ln(\hat{y}(w, x_i)) - (1 - y_i) ln(1 - \hat{y}(w, x_i)) \tag{3.23}$$

This function is in fact denoted as the *cost* or *error* function of our neural net for the task of binary classification. We have thus achieved a metric for determining the performance of our network. The task of training the network is reduced to finding the minimum of the error-function.

3.4.2 Sharpe

When evaluating the performance of an investment one usually examines the *Sharpe Ratio* which is a measure of the investments return adjusted for its risk. The task of predicting future returns of financial assets is essentially a precursor for maximizing our Sharpe ratio. The adequacy of investigating the directional accuracy relies on the fact that it *could* be used in a strategy which maximizes our Sharpe ratio. Is it possible that we could circumvent this and let our model directly optimize to maximize Sharpe? The Sharpe ratio is defined as

$$SR = \frac{E(R) - r_f}{\sqrt{var(R)}} \tag{3.24}$$

where R denotes our portfolios' return over a period and r_f the risk-free rate. How do we incorporate this as our loss function? In the case of optimizing the binary cross-entropy, we let the activation function of the output layer be a sigmoid which can be interpreted as the probability of the input belonging to a the positive class. In this case we let the activation function be the hyperbolic tangent, tanh. This scales the output to a variable $\hat{y}(x, w) \in [-1, 1]$ which can be interpreted as the size of our position with negative numbers denoting a short position. We then calculate the returns of our portfolio as

$$R_B = r_B y(x, w) \tag{3.25}$$

where B denotes our batch size. We specify our custom loss function as

$$J(w) = -\frac{\frac{1}{B}\sum R_{Bi}}{\frac{1}{B}\sum(R_{Bi} - \mu_{R_B})^2} \tag{3.26}$$

Note that we specify the loss as the negative Sharpe which we want to minimize. If directional accuracy were a good indicator of a high Sharpe ratio we would assume that when scaling our output such that $\hat{y} \in [0, 1]$, and interpreting it as a measure of class assignment, we would achieve similar accuracy to that of the binary cross-entropy method.

3.5 Stochastic Gradient Descent

3.5.1 Parameter Optimization

The process of optimizing the parameters w to find the minimum value of a function $E(w)$ is a well studied problem in optimization. Given that $E(w)$ is smooth and continuous function of w, we can find the gradient $\nabla E(w)$ which points in the direction of the greatest *increase* of the error function. The point w such that

$$\nabla E(\mathbf{w}) = 0 \tag{3.27}$$

is called a stationary point and could represent both a local minimum, maximum or saddle point. Our error function could be highly non-convex, giving us several points where the gradient vanishes. It may not be possible to find a global minimum. To find an optimal solution would thus require us to compare several local minima to find a sufficiently good solution.

We proceed by developing a method to numerically find a solution to 3.27. We want to find a scheme that iteratively updates our weights $\mathbf{w}$ such that $E(\mathbf{w})$ reaches its smallest value. One such update scheme could be described by

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \Delta\mathbf{w}^{(t)}. \tag{3.28}$$

Our main concern involves the choice of $\Delta\mathbf{w}^{(t)}$. Consider adding random noise to $\mathbf{w}^{(t)}$. That is we let

$$\Delta\mathbf{w}^{(t)} \sim \mathcal{N}(0, \sigma^2) \tag{3.29}$$

and then evaluate $E(\mathbf{w}^{(t+1)})$. We repeat this experiment k times and pick the value of $\mathbf{w}^{(t+1)}$ where E has decreased the most. We can improve this evolutionary strategy by using additional information provided by $E(\mathbf{w})$. We want to minimize $E(\mathbf{w})$, we could thus take steps in the *opposite* direction of the gradient $\nabla E(\mathbf{w})$ such that

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla E(\mathbf{w}^t) \tag{3.30}$$

This is the most basic form of the *gradient descent* algorithm where the parameter η is known as the learning rate which is a *hyperparameter* that needs to be chosen. In this case the error function E is defined w.r.t the training set. This implies that to evaluate the gradient, we need to first process the entire dataset, which could be very time consuming. Consider a dataset with N training examples $x_1, \ldots, x_N$. If we create a new dataset which is double the size by duplicating each element, the error function will simply be scaled by a constant factor which will make no difference in the gradient descent algorithm. However, this will take twice the amount of computational time to evaluate.

There exists another approach which takes this problem into account. Instead of evaluating the the error function on the entire dataset we consider it as a sum of individual errors for each training example,

$$E(w) = \sum_{n=1}^{N} E_n(w) \tag{3.31}$$

We then evaluate the gradient for each training example and update the weights as such

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla E_n(\mathbf{w}^t) \tag{3.32}$$

This version of gradient descent is called *stochastic gradient descent.* Since we perform a weight update for each data point, it is possible that we will get very noisy gradients that may lead us away from the direction of the batch gradient descent. We can instead consider a trade-off between these two approaches. Consider using a *mini-batch* of size B which is randomly sampled from the data. We then perform $\lceil \frac{N}{B} \rceil$ gradient updates at each epoch. When using mini-batch gradient descent we achieve the computational efficiency of stochastic gradient descent while not being prone to the noisiness of the algorithm. The batch-size B is a hyperparameter which needs to be optimized together with the learning rate η to find a good pair.

We note that a practical implementation of the gradient descent algorithm involves analytically calculating the gradient $\nabla E(\mathbf{w})$. This is by considering the output of our network as composite function of our weight layers and taking the derivative, using the chain rule, w.r.t to all the weight layers.

3.6 Optimizers

The learning rate fills a crucial role in the vanilla stochastic gradient descent. A small learning rate yields an update scheme which could be very slow to converge or even fail to prohibit the weights to get stuck in a non-optimal minimum. On the other hand a large learning rate could *overshoot* and fail to converge completely. It is common to adjust the learning rate at various epochs of the algorithm to avoid these problems. A simple adjustment would be to exponentially decay the learning rate at each time step, so that

$$\eta(t+1) = \alpha\eta(t) \quad \alpha \in [0,1] \tag{3.33}$$

As the algorithm reaches a minimum, the learning rate decays and can fine tune the model at a level where a large learning rate would fail. Another issue we face is that we apply the same learning rate for each parameter. If our input data contains features which change with varying frequencies we may need to make larger changes to some parameters of our model and not others. We would thus need different learning rates for each parameter.

3.6.1 Adagrad

Various adjustments to the gradient descent algorithm exists. One of them, ADAGRAD incorporates both learning rate decay and different learning rates for each parameter. Let

$$g_{t,i} = \nabla E(w_{t,i}) \tag{3.34}$$

denote the gradient of parameter w_i at time step t. We introduce the diagonal matrix G_t where the diagonal element $G_{t,ii}$ is the sum of the past squared gradients of parameter i up to timestep t. The weights are then updated as follows

$$w_{t+1,i} = w_{t,i} - \frac{\eta}{\sqrt{G_{t,ii}\varepsilon}}\nabla E(w_{t,i}) \tag{3.35}$$

This results in that parameters which are rarely updated, with a small accumulative gradient contribute to a larger step in the gradient update.

3.6.2 Adam

ADAM [13] is another update scheme which computes adaptive learning rates. Adam keeps track of both past gradients and past squared gradients as follows

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{3.36}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2. \tag{3.37}$$

It thus computes an estimate of both the mean and variance of the gradients. The initial values of m_t and v_t are 0. This implies that when the decay rates are small, i.e β_1, β_2 are close to one, m_t and v_t will be biased towards 0. To adjust for this, we compute the bias corrected mean and variance,

$$\hat{m}_t = \frac{m_t}{1 - \beta_1} \tag{3.38}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2} \tag{3.39}$$

Similar to ADAGRAD, we then compute the weight updates by,

$$_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t} + \varepsilon} \hat{m}_t \tag{3.40}$$

3.7 Regularization

Given the large number of parameters in a Neural Network, it is very prone to *overfit* the data. Overfitting occurs when the criterion in model-selection is not the same as when judging the performance of the model. When training the model, we want to fit the *training data* as well as possible by minimizing the loss function. When evaluating the model, we test it on unseen data which the model has not been optimized for. Overfitting our model refers to the fact that our model has learned the training set to well and cannot generalize well to unseen data. To prevent this we must introduce some limits on the networks capabilities.
There are three common ways to regularize Neural Networks.

1. Constraining the parameter values.
2. Ensemble learning.
3. More training data.

We will mostly be concerned with the first two.

3.7.1 L^1 regularization

A common way to regularize a statistical model is introducing limits on the magnitude of the parameters. This can be done by introducing a regularization term to the loss. Let J denote our cost function, to regularize our model we could introduce the new cost function

$$J(f(W,x),y) + \lambda R(W) \tag{3.41}$$

A common choice for R is the L^1 norm over all parameters.

$$R(W) = \sum_{k=1}^{L} \sum_{i,j} |W_{i,j}^{k}| \tag{3.42}$$

The objective function to minimize is thus the prediction error regularized on the condition that the elements of the weight matrices should not be too large. The parameter λ determines the amount of regularization we want to introduce. To much regularization can instead lead to our model underfitting the data which implies that it cannot adequately capture the structure of the data.

3.7.2 Ensemble Learning

Wisdom of Crowds is the idea that a large group of people are collectively better at decision making than even individual experts. In Ensemble Learning we apply this idea to increase the predictive performance of our classifiers by decreasing generalization error. Given a set of k trained models $\mathcal{M} = \{M_1, \dots, M_k\}$ we can combine the these into a single classifier by considering the average of their outputs.

$$M(x) = \frac{1}{k} \sum_{i=1}^{k} M_k(x) \tag{3.43}$$

This is the simple average of the model, we can consider geometric or other weighted averages as well. A further possibility is training a new classifier given the outputs of our k models to learn the optimal weighting scheme. This is called *stacking.*
Two conditions for Ensemble Learning to increase performance is that the individual performance of each classifier must be good while preserving *diversity* between the classifiers. This essentially keeps the average performance the same but decreased the variance of the decisions.

3.7.3 Dropout

Dropout [10] was introduced in 2012 as a novel way to prevent overfitting in Neural Networks. Consider a one-layer neural network with n hidden units. Dropout applies a probability p on each unit to randomly be removed from the network at each forward pass during *training*. Consider the forward pass at the l^{th} layer of a feed-forward neural network.

$$h_t^l = \sigma(W_l h_{l-1} + b_l) \tag{3.44}$$

Applying dropout to layer l coincides with introducing a Bernoulli distributed random variable $r^l \in R^d$ of the same dimension d as the previous output h_{t-1}

$$r_i^l \sim \text{Bernoulli}(p) \tag{3.45}$$

$$\hat{h}_{l-1} = h_{l-1} \circ r_l \tag{3.46}$$

$$h_t^l = \sigma(W_l \hat{h}_{l-1} + b_l) \tag{3.47}$$

This is equivalent to setting certain columns in W_l to 0.
The neurons to drop are sampled randomly at each forward pass in training. When testing, we scale each neuron by p such that the output at test time is the expected output during training. It is not immediately clear how dropout increases generalization in the network. Dropout can be regarded as a form of ensemble learning. In batch-learning, we update the weights by averaging the losses for each training example in the mini-batch. With dropout applied we are essentially creating a new, smaller network at each forward pass. During testing we use the entire collection of neurons and scale each weight by p. This is equivalent to averaging over all the models created by Dropout.

3.7.4 Recurrent Dropout

Dropout can be applied between recurrent connections as well. A recurrent unit processes information both through input-output connections as well as along the time-dimension through its states. Recurrent Dropout [6] drops neurons in the time-dependent recurrent connection only.

3.7.5 Early Stopping

When using a gradient-descent based optimization scheme, the number of epochs impacts the level of overfitting in the model. As with all statistical learning algorithms, we are fitting a model based on only a sample of the data. The optimal parameter values we attain will thus not be entirely descriptive of the true distribution. Early Stopping takes this into account and by utilizing a hold-out set or validation set and computing the loss on this set during training as well. When the validation loss stops decreasing, further optimization will lead to over-fitting, hence early stopping will halt training at this point.

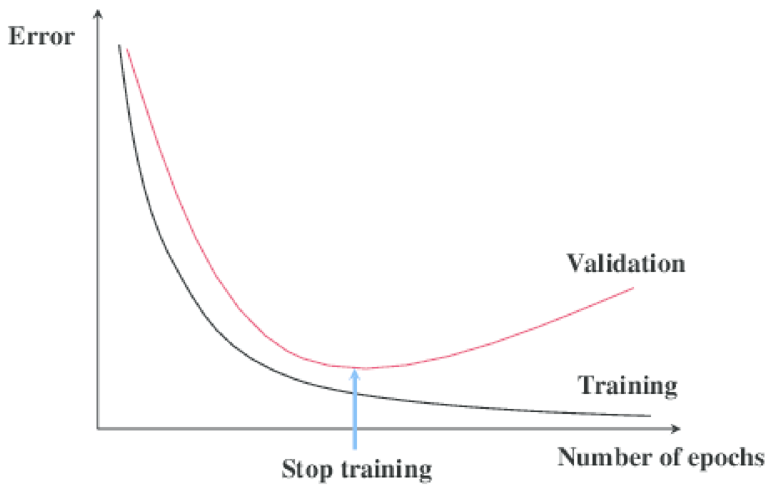


Figure 3.5: Overfitting can be defined as the point when the validation loss reaches its minimum.

3.7.6 Multi-Task Learning

When training a Neural Network we are usually concerned with only one metric, be it accuracy of a classifier or the mean-square error of a regression. There can however be different, but related metrics which could help the network in extracting signals which tell us something more about the nature of our original metric. In multi-task learning we create a network with several shared layers followed by individual layers, each optimized for an individual metric. The final loss is calculated as a weighted average over all loss-functions.

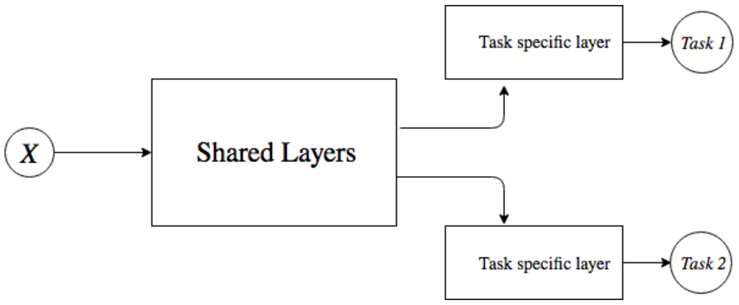


Figure 3.6: An overview of Multi-task learning. The shared layers extracts features which optimize performance for both tasks while the task specific layers only optimize for an individual task.

If we are interested in optimizing one metric, introducing another one can either work by aiding in feature extraction, or it could work as a regularizer. If the shared layers have two goals to optimize for, it won't necessarily over-fit to one of the metrics.

3.8 Scaling

Before inputting the data into the model, it is common practice to perform **feature scaling** which standardizes the range of values that each feature can attain. This is especially important if the scale of the features differ a lot. Feature scaling is important for mainly two reasons:

1. Model interpretability. With each feature scaled to the same range of values, the parameters of the model can reveal which features are most significant for the task. This is most apparent in classical Statistical methods such as Linear Regression where the absolute value of the parameters related to more important features will be larger. For our neural network the value of the weights is not necessarily related to feature importance.

2. Optimization. On of the key components of the performance of the Neural Network is on how well our learning algorithm converges. With our parameters initialized as $\theta \in [0, 1]$, unscaled features will lead to our optimization scheme beginning at a point further from a local minimum compared to the rescaled features and we run a bigger risk of ending up at a non-optimal minimum. In the gradient descent optimization scheme, the weight updates are also dependent on the value of the feature itself, leading to certain weights updating faster than others. Feature scaling thus leads to increased performance of the learning algorithm.

3.8.1 Min-Max Scaler

A common way to scale features is to use the min-max scaler.

$$\hat{X} = \frac{X - min(X)}{max(X) - min(X)} \tag{3.48}$$

This scales the input X into $\hat{X} \in [0, 1]$. The minimum and maximum are taken over the entire training set and the values are kept for the test set when re-scaling during testing. An issue with the min-max scaler is that it scales the data relative to minimum and maximum of the dataset which are *outliers*. Thus outlier features will be scaled to 1 while the rest of the dataset may be scaled to values closer to 0.

3.8.2 Robust Scaler

An alternative to the Min-Max scaler is the Robust Scaler. This alternative is a method that is more robust to outliers. It does this by considering the inter-quartile range which is the range between the first and third quartile. The robust-scaler can be described as

$$\hat{X} = \frac{X - median(X)}{Q_3 - Q_1} \tag{3.49}$$

where Q_i denotes the i^{th} quartile.
Scaling using the robust-scaler provides a more even distribution of the dataset.

Chapter 4

Methodology

Data from eight futures contract from $2000-01-01$ to $2018-01-16$ is used as input. The markets are chosen to represent FX markets, commodities and interest rates. The data consists of the open and close aswell as the highest and lowest price for each day. These are denoted O_t, C_t, H_t, L_t respectively. The data is summarized in Table 4.1.

Table 4.1: Data used for our model.

Gold	Commodity	Future
Crude	Commodity	Future
CAD	Forex	Future
EUR	Forex	Future
GBP	Forex	Future
AUD	Forex	Future
SP500	Stock Indices	Future
Tbond	Fixed Income	Future

A larger set of assets could ofcourse have been chosen. However, we want to keep the number of assets relatively low to prevent having a large set of features compared to our number of data points.

Before we dive into our methodology it is worth mentioning that all analysis was performed using Python 3.6 with the Deep Learning Framework Keras [3].

4.1 Pre-processing

Before we input the data into our model we first begin by extracting a set of *features* which we believe may be relevant towards predicting future market activity. Feature extraction and selection may be considered the most important part of our modelling task. Given a set of features which are uncorrelated and highly informative for our prediction task, the choice of model becomes less relevant. One advantage of neural networks compared to more traditional methods is its ability to automatically extract relevant features. An example is Speech recognition where traditional methods consists of preprocessing the speech signal through a series of filters which are designed similar to how humans process speech. Today, the raw signal is sufficient to serve as input to a neural network since the feature extraction is handled automatically by the network. Hence, it is also relevant to investigate if the raw closing prices C_t is sufficient to serve as input to our model. This will be compared to a set of features which are based on Widegren's [18] feature setup.

4.2 Features

4.2.1 Simple Moving Average

A simple feature which summarizes past performance is a *moving average*. The simple moving average (SMA) of the past m days is defined as

$$SMA(X_t, m) = \frac{1}{m} \sum_{i=0}^{m-1} X_{t-i} \tag{4.1}$$

4.2.2 Exponential Moving Average

The simple moving average weights all past m points equally. In theory, the more recent prices should be more relevant and thus be weighted higher in our average. The exponential moving average (EMA), gives an exponentially decaying weight α to *all* past points. It is given by

$$EMA(X_t, \alpha) = \sum_{i=0}^{\infty} \alpha(1-\alpha)^i X_{t-i}, \quad |\alpha| < 1 \tag{4.2}$$

It is common do define the EMA by a lookback period of m days similar to the SMA. This is done by choosing α by

$$\alpha = \frac{2}{1+m} \tag{4.3}$$

where m is the number of past days to take into consideration.

4.2.3 Momentum

Momentum is defined as the price difference between todays price and the price k days ago.

$$Momentum(X_t, k) = X_t - X_{t-k} \tag{4.4}$$

If the momentum is positive, this could be indicative of an upwards trend in the price.

4.2.4 Stochastic K%

The stochastic K% feature over the past K days can be interpreted as a measure of how overbought or oversold an asset is. It is given by

$$StockK(m)_t = \frac{C_t - min\{L_t, \ldots, L_{t-k}\}}{max\{H_t, \ldots, H_{t-k}\} - min\{H_t, \ldots, H_{t-k}\}} \cdot 100 \tag{4.5}$$

It compares the current closing price C_t to the relative to the past highest and lowest price.

4.2.5 Relative Strength Index

The Relative Strength Index (RSI) is another measure of the *momentum* of a stock. Let

$$\delta_t = C_t - C_{t-1} \tag{4.6}$$
$$U_t = \delta_t I_{\delta_t > 0} \tag{4.7}$$
$$D_t = \delta_t I_{\delta_t < 0} \tag{4.8}$$

. We define the Relative Strength as the fraction of the average of the upward movements U_t and average of the downward movements D_t. The RSI is then this value scaled between 0 and 100 as follows

$$RS = \frac{SMA(U, m)}{SMA(D, m)} \tag{4.9}$$
$$RSI(m) = 100 - \frac{100}{1 + RS} \tag{4.10}$$

Usually a value below 30 would indicate that the asset is oversold while a value greater than 70 is an indicator that the asset is overbought.

4.2.6 Return vs Risk

The purpose of the Return vs Risk indicator is to provide a measure for the trade-off between risk and return in the contract.

$$\hat{V}_t(m) = EMA((C_t - C_{t-1})^2, m) \tag{4.11}$$

$$RvR(m) = \frac{C_t - C_{t-1}}{\sqrt{\hat{V}_{t-1}(m)}} \tag{4.12}$$

4.2.7 Commodity Channel Index

The Commodity Channel Index (CCI) is a measure of how far the price has strayed from its moving average in terms of a moving average of the mean deviation. The Commodity Channel Index is thus defined as

$$X_t = \frac{C_t + H_t + L_t}{3} \tag{4.13}$$

$$CCI(m) = \frac{X_t - SMA(X_t, m)}{0.015 \cdot SMA(|X_t - SMA(X_t, m)|, m)} \tag{4.14}$$

4.2.8 Percentage Price Oscillator

The Percentage Price Oscillator (PPO) is a measure of the relative difference between two moving averages.

$$PPO(m_1, m_2) = \frac{SMA(X_t, m_1) - SMA(X_t, m_2)}{SMA(X_t, m_2)} \cdot 100 \tag{4.15}$$

4.2.9 Williams % R

The relation between the current price compared to the high and low can reveal information of about volatility and if the asset is overbought or oversold. The Williams % R is defined as

$$WilliamsR(m)_t = -\frac{max\{H_t, \dots H_{t-m}\} - C_t}{max\{H_t, \dots, H_{t-m}\} - min\{L_t, \dots, L_{t-m}\}} \cdot 100 \tag{4.16}$$

It is bounded in the range $[-100, 0]$. A value in $[-20, 0]$ indicates that the asset is overbought while a value in the range of $[-100, -80]$ gives an indication of how oversold the asset is. A common value m is 14.

4.3 Feature Setup and further processing

To evaluate feature importance we will mainly compare three different feature setups. We will use a simple setup which consists mainly of the difference between various exponential moving averages of various time lengths. We will also consider a more complex setup which will take into account both a larger feature setup and also more advanced features which should give a more detailed account for the current market activity. We will also consider using just the lagged closing price ∇C_t as the only feature. A detailed account for the features is given below. All moving averages are calculated on the closing price C_t.

4.3.1 Simple Features

- $C_t - EMA(10)$
- $C_t - EMA(20)$
- $C_t - EMA(60)$
- $EMA(5) - EMA(60)$
- $EMA(20) - EMA(60)$
- $EMA(40) - EMA(60)$

4.3.2 Complex Features

- $C_t - EMA(20)$
- $C_t - EMA(60)$
- $EMA(5) - Momentum(10)$
- $EMA(5) - Momentum(20)$
- $EMA(5) - Momentum(60)$
- Stochastic K% with parameter 14
- RSI with parameter 14
- Percentage Price Oscillator with parameters $(5, 10)$
- Return Vs Risk with parameter 20
- Williams Percent R with parameter 14
- Commodity Channel Index with parameter 20

4.3.3 Close features

- $\nabla C_t = C_t - C_{t-1}$

4.3.4 A note on stationarity

The stationarity property was outlined in section 2.3 and defines the conditions that enable classical time-series models to function properly. The property of equal distribution of our input data across the dataset aids our neural networks as well. Consider just the closing price C_t as an input. The test set could possibly exhibit higher or lower closing prices than the network has encountered before, thus prohibiting the network to make educated guesses for the output. By differencing C_t, we hope to map our input data to a bounded set that is equally distributed across time, since the price jump between each day should be similar across the dataset. This could however still not be a stationary set, particularly if our test set contains the financial crisis of 2008. This issue is manifested in all our features and could be further prohibited by considering the percentage change instead of the absolute difference for some features. The advantage of our features as they are now is that they are based on classical market indicators and provide an explanatory power in our analysis.

4.3.5 Bias

One of the most challenging aspects of modeling financial data is its low signal-to-noise ratio. The inherent noisiness in the data makes any model prone to reducing the parameters to 0 and being affected only by a constant bias term. To counteract this, it is common to remove any bias terms in the model. This may however not be enough. Even when removing the bias term, our model still converged to predicting a constant up or down movement throughout the test data. This is due to the fact that if a feature is always positive, the network can itself create a bias term from this data which again leaves us with the same problem we started with. Scaling by removing the mean forces features to never be always positive and removes any bias in the model.

4.3.6 Data splitting

Scaling is performed using the Robust-Scaler as described in Section 3.8.2. The data is split into training, validation and test splits which refers to a $60\% - 20\% - 20\%$ split of the data respectively. Our training, validation and test set is thus composed of the following dates.

Table 4.2: Dates for our data split.

	Start	End
Train	2000-01-01	2011-07-21
Validation	2011-07-22	2014-06-16
Test	2014-06-17	2017-12-27

Because we are dealing with time-dependent data, the order of the data is kept during splitting. We will also feed the data chronologically into our models to further avoid any look-ahead bias when training.

For the recurrent architecture our inputs are sequences instead of individual data points. There are several ways to reshape the data into sequences. A common way is to consider a moving window of sequences of a fixed size k. The input data X will thus have dimensions $X \in \mathbf{R}^{N' \times K \times D}$ with $N' = N - k + 1$. However, this method disregards the fact that the RNN may want to take into account events which are further back then k data points in time. We can instead consider inputting the entire sequence into the model and evaluating each prediction $\hat{y}_t$ against the true value y_t at each time-step. This is done by splitting the data X into sequences of fixed size k which are *non-overlapping*. We process the $\lfloor \frac{N}{k} \rfloor$ sequences by initializing the *state* of the RNN with the state of the previous sequence. Thus the memory of all past events is preserved throughout the training. This is done practically in Keras by setting the parameter STATEFUL = TRUE in our recurrent units.

At the beginning of the prediction, we ideally want to initialize the state of the RNN. We do this by providing the RNN with a *burn-in* period to let it initialize the states. At each epoch, the state is thus initialized through a forward-pass of the first k elements in the sequence. This aides the RNN in providing more accurate predictions from the beginning.

4.4 Metrics

This thesis is concerned with two objectives. The first one is to evaluate to overall performance of a feed-forward neural network compared to various recurrent architectures. The metrics which we will utilize to compare models will be the directional accuracy of the classifiers together with the area under the Receiving Operating characteristic (ROC) curve, denoted as ROC-AUC.

The ROC curve is created by measuring the True positive rate (TPR) compared to the False positive rate (FPR) of our model when we vary the threshold probability in our binary classifier. The ROC-AUC is then the area under this curve which coincides with the probability of predicting a true positive compared to a false positive. Consider a model which only predicts upward movements in our time-series. If the up and down movements are unbalanced, we could possibly achieve a high accuracy using this method. We will thus achieve a true positive rate of 1. Varying the threshold to only predicting downward movements gives a TPR and FPR of 0. Hence the ROC-AUC will be the area under the triangle created from the straight line from $(0, 0)$ to $(1, 1)$, which is 0.5. Hence, the ROC-AUC is a more significant performance metric than the accuracy.

The second problem investigates the benefit of these metrics when put in the context of creating an automated trading algorithm. A simple strategy which we will investigate is to trade everyday based on the next-day prediction of an up or down movement in the market. The metric used to evaluate this strategy is the Sharpe Ratio. Good predictions on the direction of the market has clear benefits but it is not certain that they directly translate to a high Sharpe ratio. This discrepancy will be examined by training networks optimized either for Accuracy or Sharpe and evaluating them on both of these metrics. We will further investigate this by considering multi-loss networks which optimize for both objectives at the same time.

4.5 Architectures

We will consider a wide range of feed-forward and recurrent architectures. We want to investigate the importance of the number of hidden layers as well as layer size in the model. This is done in part to investigate if larger networks provide an advantage when training our network on certain feature setups. Another reason is that performance of neural networks is highly dependent on different combinations of its hyperparameters. The number of layers together with the number of nodes of each layer has an impact on the optimal learning rate and batch size. This symbiotic relationship impels us to consider several combinations of hyperparameters to be able to provide a fair comparison when studying our research questions.

4.5.1 Feed-forward architectures

We will consider networks with a hidden layer depth ranging from one to three. The number of neurons in each layer will also vary and we will specifically consider layer sizes as outlined in Table 4.3

Table 4.3: Overview of Feed-Forward Networks considered.

Layers	Neurons
1	[100], [200]
2	[32,16], [64,32], [128,64]
3	[32,32,16], [64,64,32], [128,128,64]
Activation	relu
Optimizer	ADAM
Residual Connections	Yes/No
Loss function	Sharpe / Binary Cross-entropy
L1 Regularization	0.01
Batch Normalization	Yes
Dropout	0.5
Input size	80 / 160

4.5.2 Recurrent architectures

We will experiment with single layer Recurrent networks as well as stacked recurrent networks which processes the data through several different recurrent layers. The architectures are summarized in Table 4.4

Table 4.4: Overview of Recurrent Network Architectures

Layers	Neurons
1	[64], [128]
2	[32,16], [64,32], [128,64]
3	[32,32,16], [64,64,32], [128,128,64]
Optimizer	ADAM
Residual Connections	Yes/No
Loss function	Sharpe / Binary Cross-entropy
Recurrent Dropout	0.2
Recurrent Unit	[LSTM, GRU, Simple RNN]

4.6 Feature Importance

When evaluating the efficiency of a statistical learning algorithm one usually considers the accuracy of its predictions. For a researcher, one of the most important parts of a model is however its interpretability. Neural networks are very hard to interpret and solving this problem is one of the biggest challenges the research community must solve before neural networks can achieve widespread adoption throughout society. We propose a novel method for investigating the importance of the features in our model. This gives us further insight into what technical indicators have the most predictive power.

Let T denote the number of timesteps in a series. Let D denote the number of features per contract and C the number of contracts. Our input X is then of the form $X \in \mathbb{R}^{T \times DC}$ together with outputs $y \in \mathbb{R}^{T \times C}$. For each timestep $t \in T$ and market $c \in C$ consider the partial derivatives

$$J_{i,j}^{t,c} = \frac{\partial y_{t,c}}{\partial X_{t_i,d_j}} \quad t_i \in [0,t], d_j \in [cD, (c+1)D] \tag{4.17}$$

For a contract c we consider only the features X which were generated from this the time series of this contract. The implicit belief is that these features will be the most relevant when measuring the predictions on contract c. There are of course relationships between the markets which is left out from this analysis. The gradients are computed after training is done by considering inputs and outputs from the test set.

The derivatives gives us insight into how the output at timestep t is affected when slightly adjusting the values of each feature at previous timesteps. Hence, a hypothesis is that the most important features will have the largest absolute gradient, since the networks output is more sensitive to a change from these. Furthermore, by considering all timesteps t we can investigate how far back in time the LSTM remembers. Inputs further back in time will likely have lower impact than more recent inputs.

We conduct two types of analysis using the gradients. We consider the gradients of the last data point in our series with respect to all previous input. This gives us an not only which features affect contract c but also which points in time the recurrent architectures take into account.

$$J_{i,j}^{T,c} = \frac{\partial y_{T,c}}{\partial X_{t_i,d_j}} \quad t_i \in [0,T], d_j \in [cD, (c+1)D] \tag{4.18}$$

To give a summarizing picture of the effect of the gradients, we will also consider the mean absolute value of the gradients throughout all time-steps. That is we find the absolute gradient with respect to feature j as

$$G_{T,cj} = \frac{1}{N} \sum_{i=1}^{N} |\frac{\partial y_{T,c}}{\partial X_{t_i,d_j}}| \tag{4.19}$$

This gives us a picture of the overall impact of a feature.

Chapter 5

Results

In this chapter we present the results of our findings. To ease notation, we will describe the model architectures with the following notation.

Table 5.1: Notation

	1L1	1L2	2L1	2L2	2L3	3L1	3L2	3L3
Recurrent	[64]	[128]	[32,16]	[64,32]	[128,64]	[32,16,16]	[64,32,32]	[128,64,64]
Feed-forward	[100]	[200]	[32,16]	[64,32]	[128,64]	[32,16,16]	[64,32,32]	[128,64,64]

5.1 Benchmark

We will compare our results to a buy and hold strategy as well as a naive strategy where the next prediction equals the previous direction such that if $\hat{Y}_t$ denotes our prediction at time t, we have

$$\hat{Y}_t = Y_{t-1} \tag{5.1}$$

The performance of these two strategies is presented below

Table 5.2: Benchmark strategies

	Sharpe	Return	Accuracy	ROC
Buy and Hold	-0.036	-1.7%	51,8 %	0.5
Naive strategy	-0.45	-6,3%	48,6%	0.48

5.2 Feed-forward Neural Network

5.2.1 Vanilla Feed-forward Network

We compare various feed-forward network architectures and objective functions. We also investigate the effects of including residual connections on these networks.

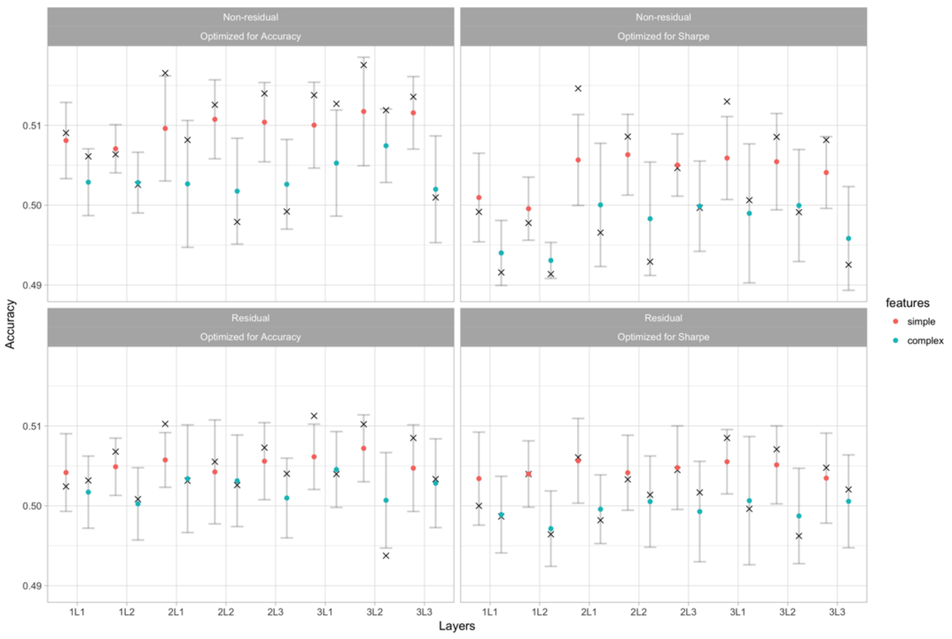


Figure 5.1: Evaluating the accuracy of our feedforward Networks. The different figures indicate whether residual connections were used or not together with which objective functions the networks were trained for.

Figure 5.1 evaluates the accuracy of our vanilla feed-forward models on our test set. Beginning with the two types of feature setups, the simple features seem to outperform across all network architectures and optimization goals. The complex features include both a larger feature setup which may benefit the network by providing more information about the time series and a set of more advanced features that should give additional insights into market behaviour. These attributes can however at the same time be detrimental to model performance since additional features introduce the probability of the model overfitting, especially with our rather limited number of datapoints. The Complex features also exhibit *quicker* features which on one hand may be more impactful when deciding the next day accuracy but at the same time can fail to capture longer term trends which may be features that are important for predicting the directional movements of the time series.

By introducing more parameters into our model by increasing both layer depth and including more neurons per layer, we can detect a slight improvement in performance. This adheres to the hypothesis that financial time series exhibit highly non-linear dependencies and thus our network requires deeper architectures to be able to make decent predictions.

Residual connections seem to exhibit lesser performance than its counterpart. The advantage of residual connections is the ability to utilize deeper networks without impacting network performance by introducing issues such as vanishing or exploding gradients. The residual connections seem to have one advantage noticeable across all architectures and that is stability. With the residual connections, each network, no matter what initialization, has the same input propagated throughout the network. We see less difference between the networks performance since we only need to learn an easier residual function $F(x)-x$ instead of each network learning $F(x)$. This introduces smaller differences between the networks which leads to more stability. We can summarize the above two discussion in Figure 5.2.

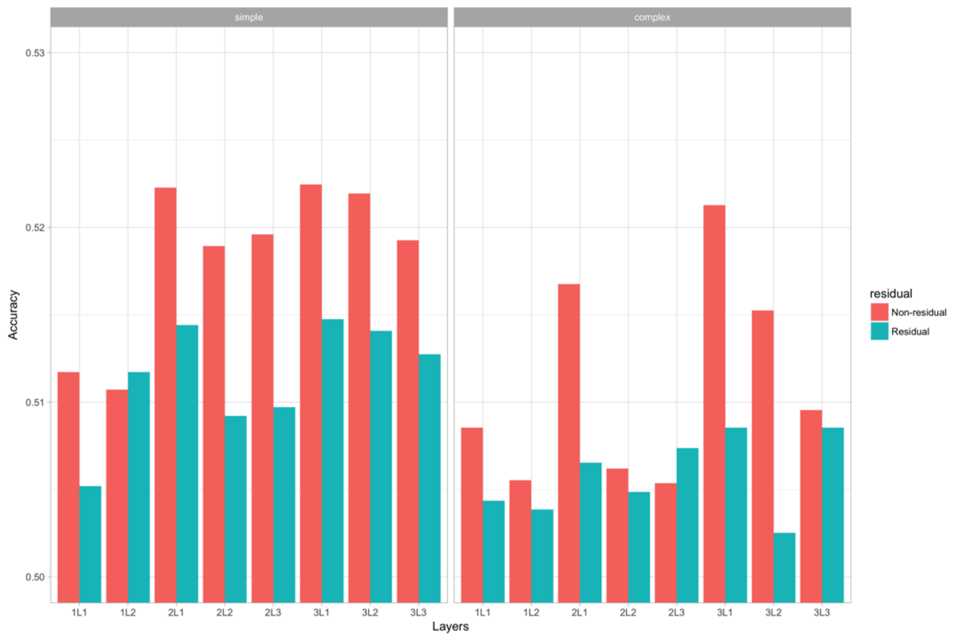


Figure 5.2: The accuracy of our ensemble grouped by number of layers and use of residual connections. More layers and non residual connections seem to be a winning combination. The figure also highlights the difference between the simple and complex features. Each model is chosen based upon its optimal combination of learning rate and epochs to give a fair comparison.

We can evaluate the accuracy when optimizing for the Sharpe Ratio by scaling the output of our model to the range $[0, 1]$ and computing the accuracy. There is no obvious reason why this would work better than the accuracy method but it could confirm the validity of a strategy based on directional accuracy. When optimizing for the Sharpe Ratio, if we were able to achieve a high accuracy, this could imply that the network itself figures out the strategy of only predicting up and down movements and acting on that information. Clearly, this is not the case of the Feedforward network. We can approach this same problem by instead evaluating the Sharpe ratio for both methods and comparing.

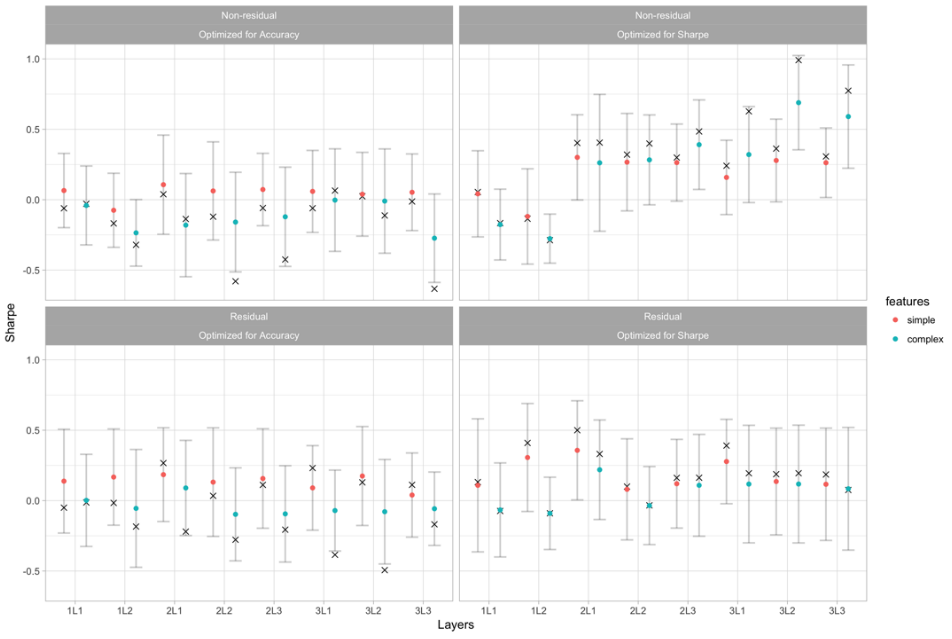


Figure 5.3: Evaluating the Sharpe of our Feedforward Networks for different objective functions and whether we include residual connections.

Not surprisingly, we achieve a higher Sharpe for networks that are also optimized for this task. Inspecting the second column in Figure 5.3 provides us with two important insights. The first is that networks with more parameters are favored compared to smaller networks. This could indicate that maximizing the Sharpe function, balancing the trade-off between return and variance, requires additional non-linear dependencies that smaller networks are incapable of capturing. A second interesting aspect of the results is that the networks optimized for Sharpe exhibit less difference in performance when utilizing either the Simple or Complex features and in the case of larger networks, favors the Complex features. The more advanced feature setup gets its name from applying features which supposedly paint a more detailed picture of the time series. To make an adequate maximization of the Sharpe Ratio we need both an accurate prediction of future market movements to maximize our return but also future predictions of market volatility, which would allow us to shrink our positions when needed to protect ourselves to market risk. The complex features especially may help us with the second problem since some of our features also incorporate past standard deviation of returns. Figure 5.4 shows the results when examining the Sharpe ratio of various ensembles optimized for Sharpe.

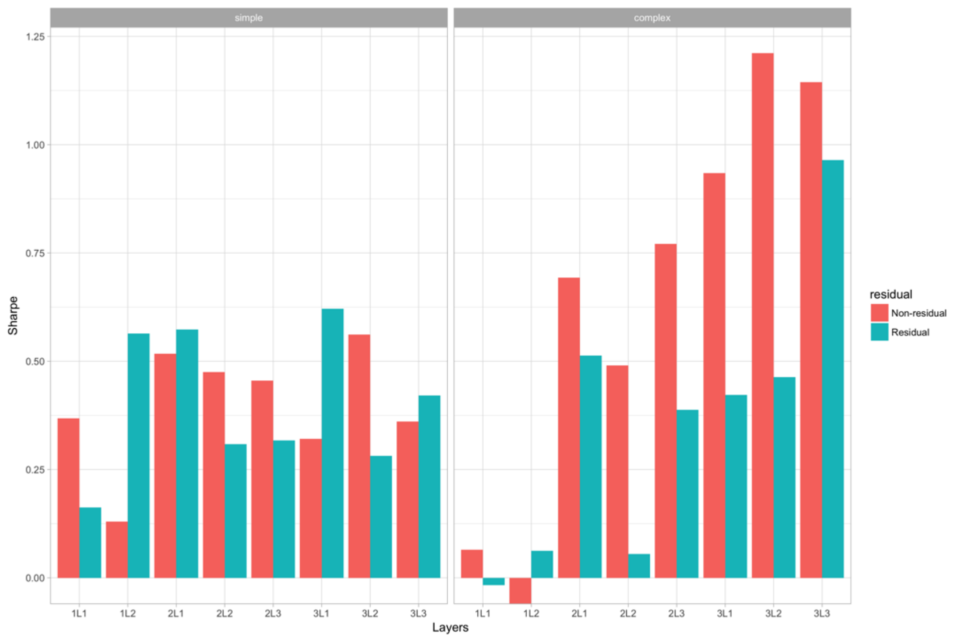


Figure 5.4: A further look into the results of networks optimized for Sharpe. Clearly complex features and non-residual connections are favored.

5.2.2 Multi-loss Feedforward Network

We examine the effect of incorporating multiple losses into our network and optimizing for both at the same time. The losses affect the weights in the shared layers by the parameter LOSS-WEIGHTS which determines how the losses are weighted when evaluating the combined loss, thus determining the importance of the loss when calculating the gradient. We plot the average score together with its error-bars of 1 standard deviation together with the results of an ensemble of networks initialized with different random seeds.

5.2.2.1 Results - Accuracy

In figure 5.5 we examine several different neural architectures and evaluate the performance of the accuracy layers given various combinations of loss weights. We evaluate the performance by examining both the accuracy and ROC-AUC of the sub-network.

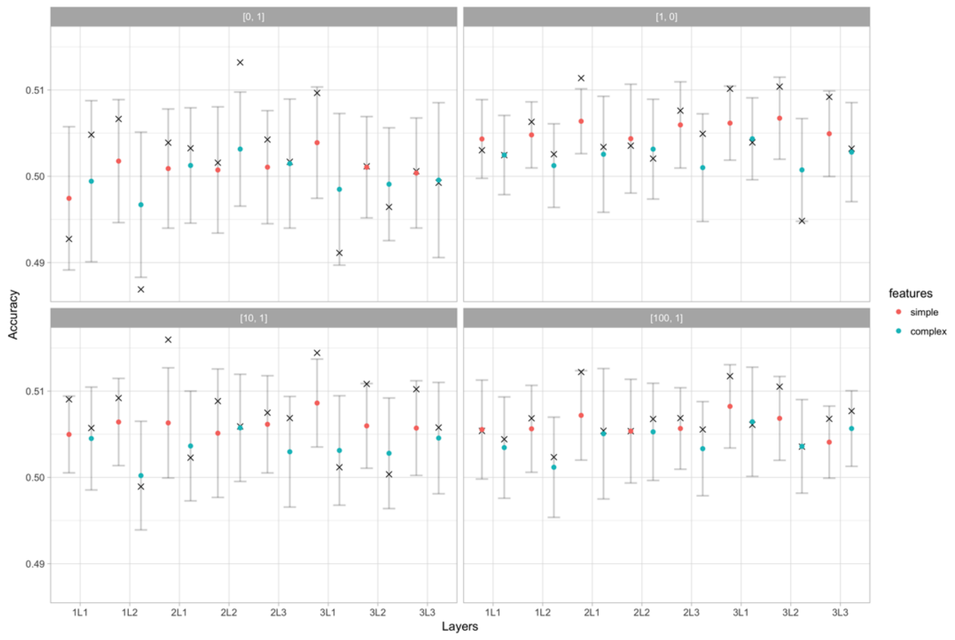


Figure 5.5: Results for the accuracy layers of our Multi-loss Feedforward Network. The different figures indicate different weights for the losses. The X-axis denotes the number of layers and layer size.

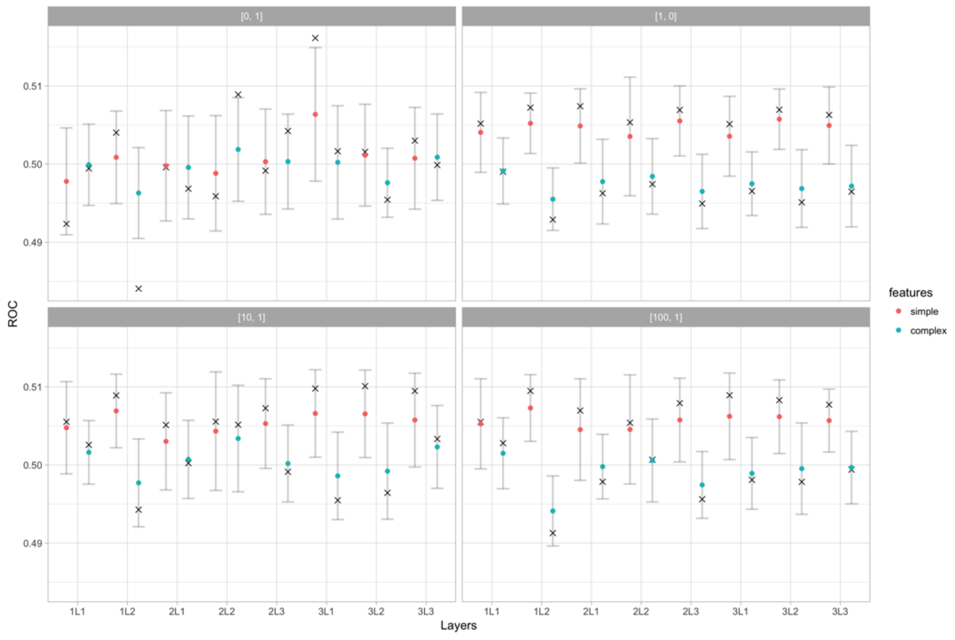


Figure 5.6: ROC-AUC for accuracy layers in our multi-loss NN.

The top-right figure gives zero weighting to the Sharpe loss function and is thus only optimized for accuracy. In the bottom figures we incorporate a small part of the Sharpe loss by weighing the loss functions with a ratio of $10:1, 100:1$ respectively towards the accuracy loss. We can note that the results become slightly better compared to only incorporating the the accuracy loss. This suggests that the regularizing effect of the multi-loss network has a positive effect on performance. As

5.2.2.2 Results - Sharpe

Next we examine both the Sharpe and Return for the layers optimized to maximize Sharpe. The figures are presented as in the previous section.

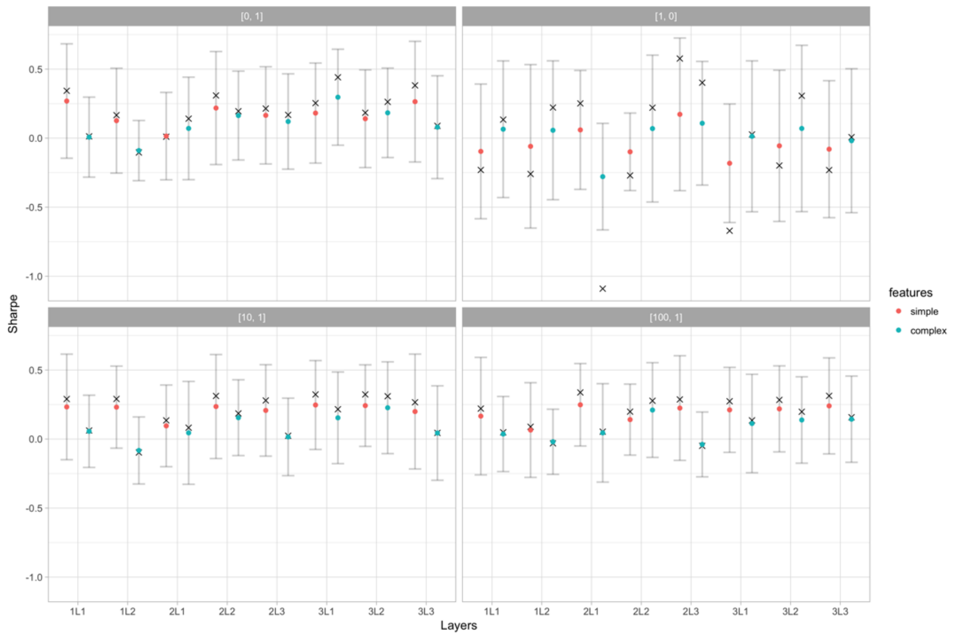


Figure 5.7: Results from evaluating the Sharpe of the layers optimized for the Sharpe loss in our Multi-Loss Network. We examine the results under different weightings of the loss functions where the weights correspond [accuracy, sharpe] respectively.

The top-left figure corresponds to optimizing directly for Sharpe and not taking into account the binary cross-entropy loss. We can compare this to incorporating the accuracy loss into the model. We note that the average results are slightly similar with perhaps a slight edge to the alternative of optimizing for Sharpe only. We can however identify that incorporating multiple losses provides slightly more stable results in certain cases. Furthermore, there is similar performance across various layer depths and sizes with notable outliers being the larger of the two-layer and three-layer networks which show worse performance with the complex features.

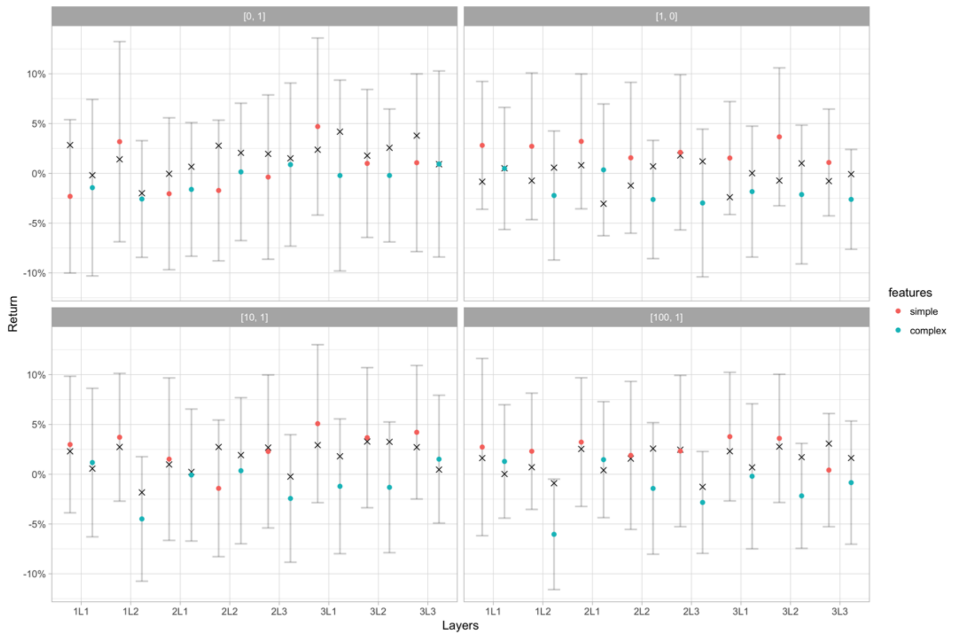


Figure 5.8: Results from evaluating the Return of the layers optimized for the Sharpe loss in our Multi-Loss Network. We examine the results under different weightings of the loss functions where the weights correspond to [accuracy, sharpe] respectively.

The return on our portfolio when optimized for the Sharpe ratio tell a different story. Comparing our results, we find that regularizing our network with an additional loss function yields both a higher average return as well as a set of more stable results.

5.3 Recurrent Neural Network

5.3.1 Comparison between different types of RNNs

We will evaluate the performance of the three different types of Recurrent Networks by comparing different optimization functions together with evaluating them on different performance metrics. Figure 5.9 shows the results when optimizing for either Sharpe or Accuracy and evaluating the accuracy of both methods. Note that in the case of optimizing for the Sharpe ratio, the positions that serve as output of the model are re-scaled to be in the interval of $[0, 1]$ which is then used to evaluate the accuracy.

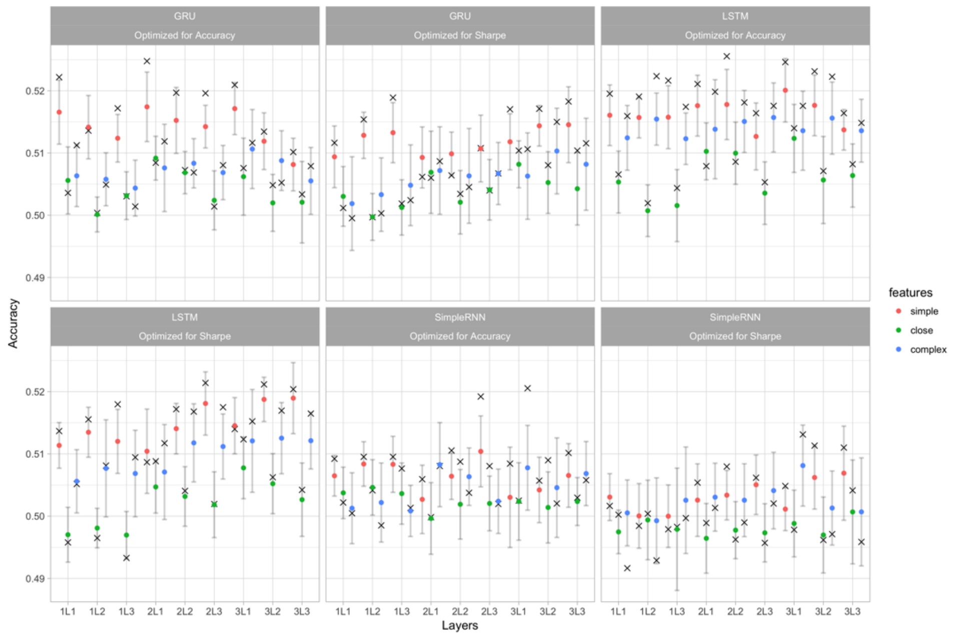


Figure 5.9: The accuracy of various RNNs with different objective functions and different feature setups.

Judging from 5.9, the simple features show superior predictive performance on the test set compared to other feature setups across all different RNNs and objective functions. As expected, when utilizing only the lag-differenced closing price, the networks perform significantly worse. The value of the average accuracy is amplified when utilizing an ensemble of models with different random seeds, which shows the power of using ensemble methods. Unsurprisingly, optimizing a model for Sharpe delivers worse performance when evaluating the accuracy. It is however notable that given the simple features, we can still attain a relatively high accuracy with the Sharpe objective function, as is most apparent in the case of GRU and the LSTM. This serves as further evidence of the intricate relationship between directional accuracy and the Sharpe Ratio.

Conversely, we can investigate the Sharpe ratio for our various model setups.

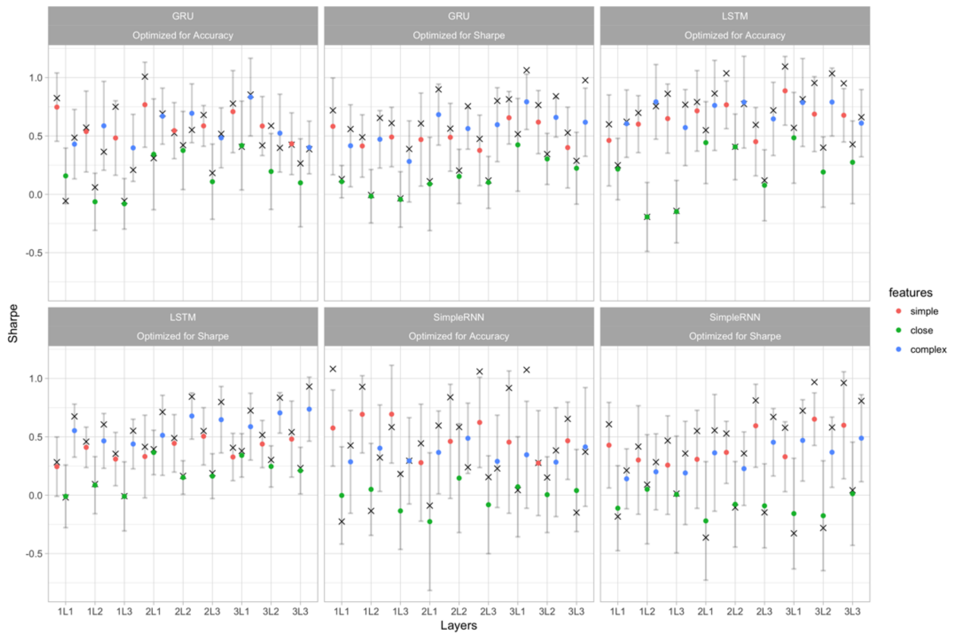


Figure 5.10: The Sharpe ratio of various strategies generated by our RNNs with different objective functions and different feature setups.

Comparing the different types of RNNs, we can draw the conclusion that the LSTM gives both a higher average Sharpe as well as slightly more stable results. Interestingly, the complex features give similar or in the case of the LSTM optimized for Sharpe, greater results than the simple features. Clearly the complex features paint a clearer picture of the optimal positions to take to maximize your Sharpe. This could be due to the fact that the complex features include features such as the Commodity Channel Index, which incorporates the mean standard deviation of the asset price, which is an intricate component when calculating the Sharpe ratio. We can further comment on the number of parameters in the model. Inspecting the models that were optimized for Sharpe, we can detect a slight upward trend in the Sharpe as the number of layers and neurons increase. This could indicate that optimizing for the Sharpe ratio is a harder problem than optimizing for accuracy. This in turn calls for more convoluted model architectures which can sufficiently extract the features that allow the model to optimize its positions to both maximize its return while keeping the variance of returns low.

The antithesis of our LSTM is the vanilla RNN which depicts a sad story of uncertainty in predictions and lesser scores. It is not however completely impractical, because it reveals an interesting aspect of our feature selection. Contrary to the GRU and LSTM, the Simple RNN favours the simple features. When impeding a models ability to look further back in time and draw more advanced conclusions, the model settles for the simple features. One of the main differences between the two feature setups is that the complex features is a collection of *quicker* features which incorporate only the past 20 to 10 days. The simple features are however composed of a set of rather slow moving averages. If we were given only a single data point, the simple features give a more comprehensive picture of current trends in the time series than the Complex features. The complex features thus require the Recurrent Network to figure out these relationships by itself, which on one hand requires more advanced architectures but at the same time proves more fruitful for our objective.

We can further investigate the Sharpe Ratio by examining the return of our generated strategies.

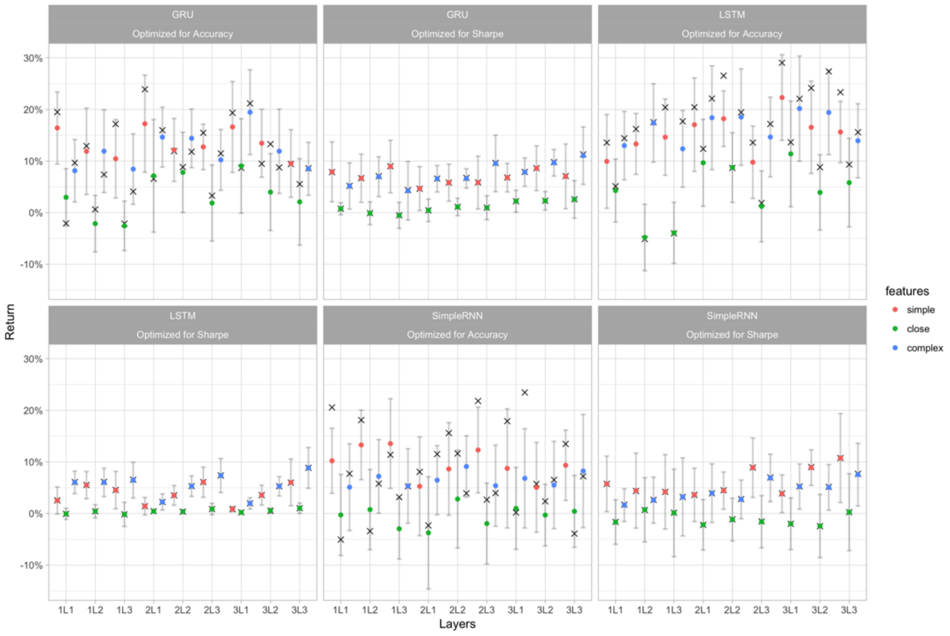
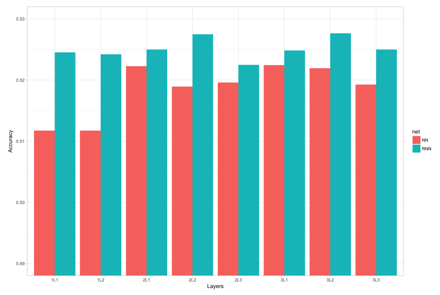


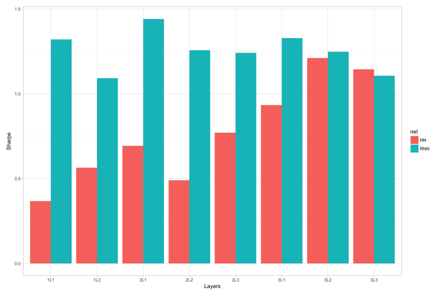
Figure 5.11: The cumulative return of various strategies generated by our RNNs.

Two immediate observations from figure 5.11 is that when using the Sharpe ratio as the objective functions, there is very little variance in the average cumulative return. As a consequence of this, we gain very little from our ensemble. We can draw one further conclusion and that is about the nature of the optimization that the networks perform. In Markowitz Portfolio Optimization, one aims to maximize the return of a portfolio while keeping its variance low. This can be done using two approaches. One can either specify a trade-off parameter between the expected return and variance which specifies our aversion to risk. A second approach has a slightly different objective. To find an optimal combination of weights for a set of assets, one can aim to minimize the variance with the secondary condition that the expected return is kept constant. As evidenced from Figure 5.11, the latter approach is exactly what our network seems to optimize for.

We can finalize our comparison of feed-forward and recurrent networks in Figure 5.12



(a)



(b)

Figure 5.12: To provide a comparison between feed-forward and recurrent networks we present the results of the best ensembles for each type of network.

5.3.2 Multi-Loss Recurrent Networks

We can now move on to investigate the effect of introducing auxiliary loss functions into our recurrent networks.

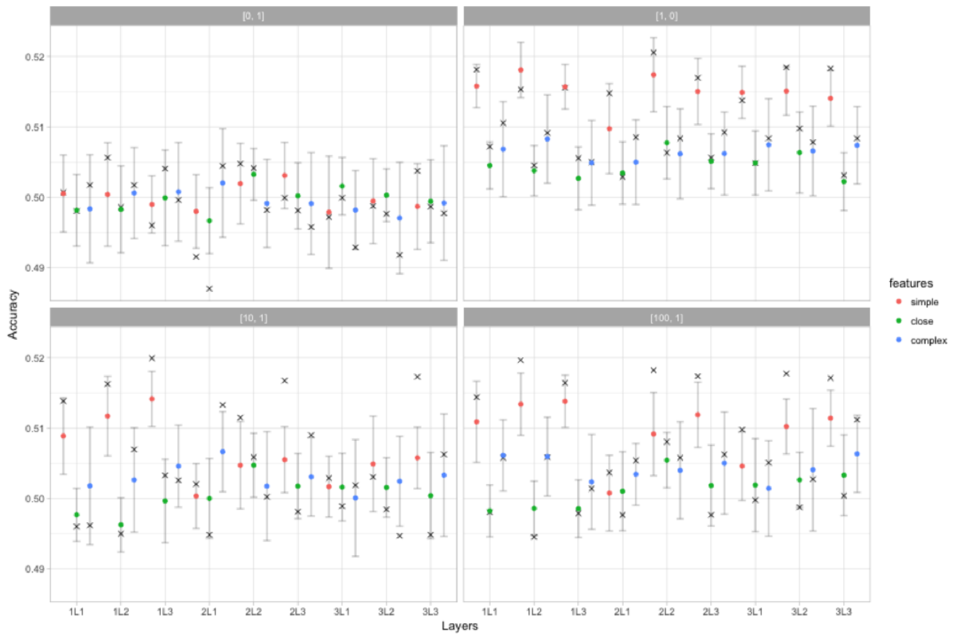


Figure 5.13: The accuracy of our multi-loss LSTM with various weighting schemes of the two different loss functions. A weight of $[1, 0]$ corresponds to only incorporating the binary cross-entropy loss.

It is clear that the regularizing effect of multi-loss networks does not work in our favor and instead allows our network to underfit the data instead. We can look at the average result of our ensembles for each weighting of the loss functions in Figure 5.14

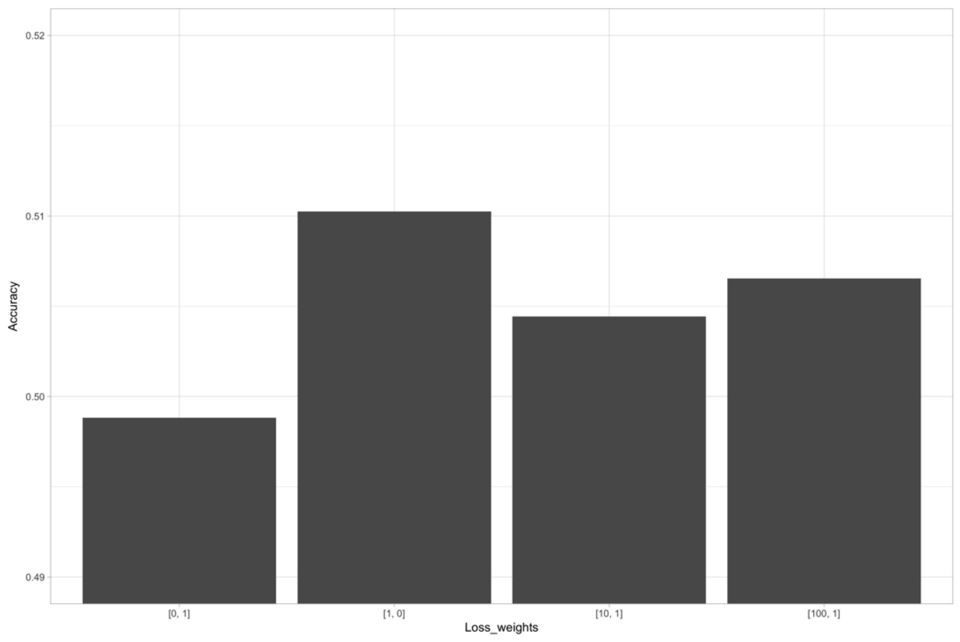


Figure 5.14: Summarized results of different weightings in the multi-loss LSTM

The effect of under-fitting becomes even more apparent as it is clear that weighing the network towards placing larger importance on the binary cross-entropy loss gives increased performance on our test set. In general, when increasing layer size or the number of model parameters, the effect of over-fitting becomes more apparent. The converse must then hold as well.

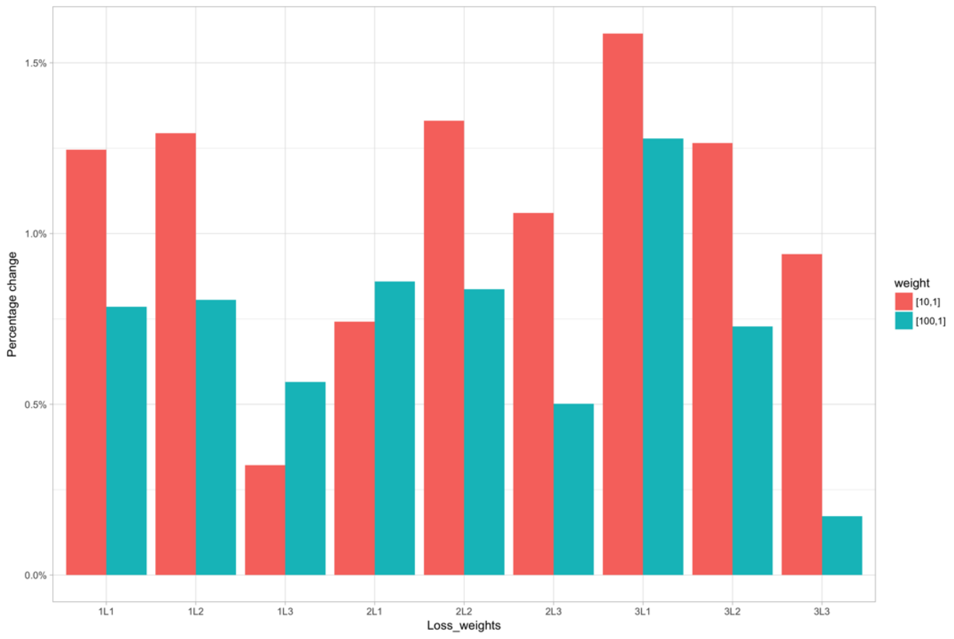


Figure 5.15: Relative difference of introducing an auxiliary loss functions compared to only incorporating the binary cross-entropy loss.

Figure 5.15 shows the difference in performance for each layer size and loss weight compared to the case of only incorporating the accuracy loss. As before, it is apparent that a weighting of $[100, 1]$ succeeds the lesser weighing. It is most notable however that the difference becomes lesser when introducing more neurons to our layers. This suggests that deeper networks with more parameters benefit more from this type of regularization .

Next we turn to examining the multi-loss regularization method on the layers optimized to maximize the Sharpe ratio.

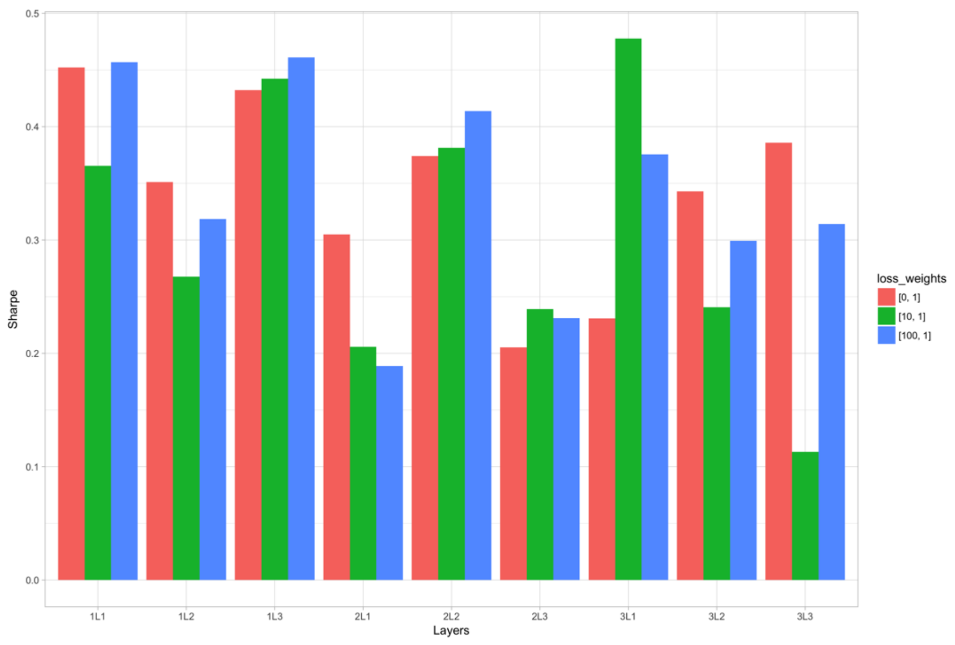


Figure 5.16: The average Sharpe ratio for various networks optimized with different loss weights.

Optimization for the Sharpe ratio benefits clearly from introducing auxiliary loss functions. Figure 5.16 presents the results of the average Sharpe ratio for networks optimized for our various loss weights. As evident in the figure, networks that do not take the binary cross-entropy loss into account struggle compared to the other networks. We can draw two conclusions from introducing another loss function into this network. We are either gaining better performance due to preventing over fitting in our network by in some way limiting the values of our weight layers, similar perhaps to an L1 regularization. Another hypothesis highlights the connection between Sharpe and directional accuracy. As for our simple networks in the previous section, evaluating the Sharpe for a layer optimized for accuracy and vice versa yielded surprisingly positive results. Another way to put this is that we essentially took all the features extracted from an Accuracy layer, which boils down to an integer $y \in \{0, 1\}$, and transformed it to evaluate the Sharpe Ratio. When introducing auxiliary losses, we are generalizing this process by letting the network incorporate a learned subset of features from the accuracy layers that aid in the process of maximizing Sharpe. One can theorize that if directional accuracy were the best way to attain a high Sharpe, our multi-loss network would learn to maximize the accuracy and the output of our Sharpe layers would simply be the re-scaled outputs of the accuracy layers, as we manually did before. This is not the case here, but we are clearly seeing an increased Sharpe by introducing auxiliary losses.

5.4 Gradient Investigation

Feature selection is perhaps the most important part of any statistical learning algorithm. The advantage of neural networks is its ability to naturally extract relevant features from the input and even generate custom-built features specific for the task. A problem with neural networks is however the limited ability for a researcher to interpret which features the network selects to be relevant to the task. We will approach this problem by evaluating the gradients of our network with respect to different features. Based on our previous results, we choose a subset of models which perform the best at their respective task and investigate the gradients of these models.

5.4.1 Memory of a Recurrent Unit

Gradients allows us to get an idea of which features are the most important for our model. Besides our simple and complex features, the recurrent networks operate on a second dimension as well, time. This is a feature that can be examined in the same way as our other inputs. Figure 5.17 shows the gradients of the output at the last time step in the AUD futures contract with respect to all previous inputs for a model optimized for accuracy.

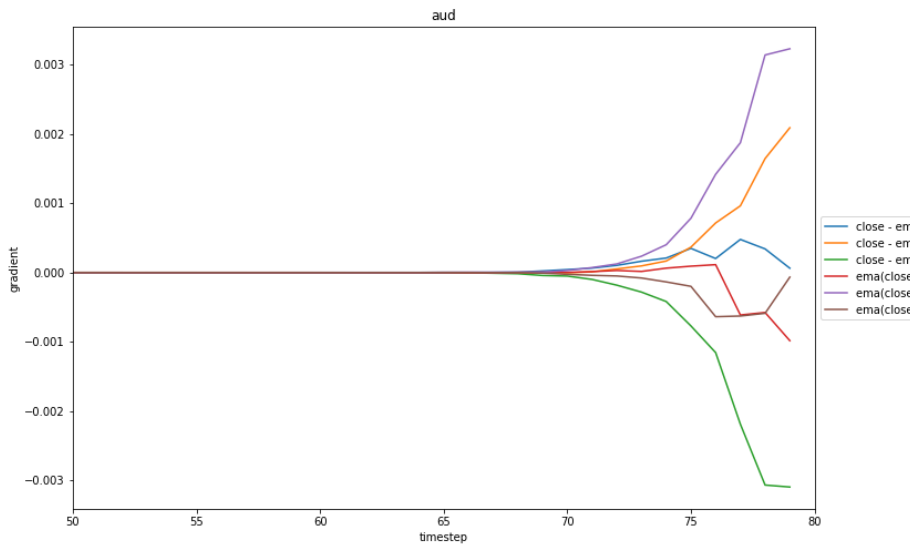


Figure 5.17: Overview of how our simple features impact the models predictions on the Australian Dollars Futures.

The figure provides an overview of which features impact the model the most, and how far back in the time the network seems to remember. From the figure, only the past 10 days seem to have any influence on the models prediction with the past 5 days being extra significant. The feature importance and time dependency varies across models, contracts and loss functions. To highlight our results we can examine the average results across similar models and feature setups. We will consider only models optimized for accuracy since we mainly want to highlight the different time dependencies of our models.

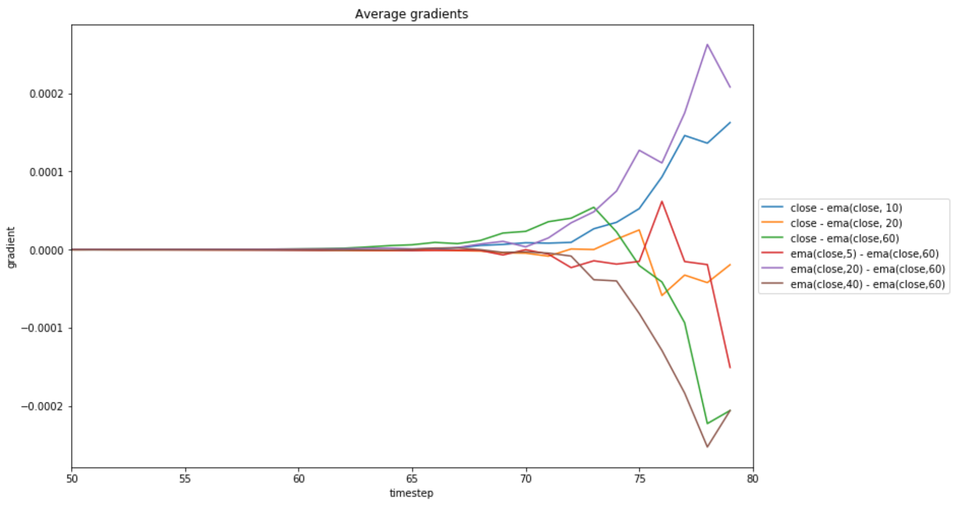


Figure 5.18: The average impact of our simple features across all LSTM models optimized for accuracy.

We can compare the memory of a LSTM to that of a simple RNN by considering Figure 5.18 and Figure 5.19.

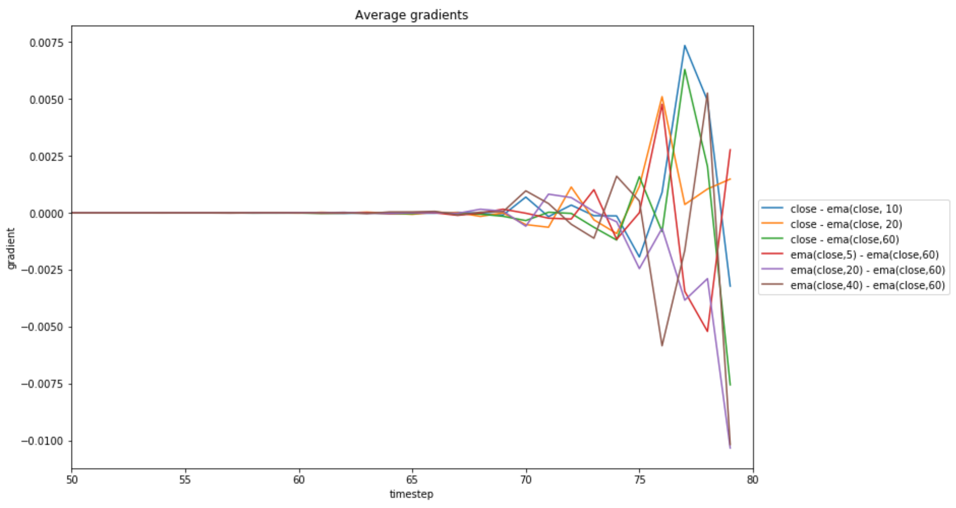


Figure 5.19: The average impact of our simple features across our simple RNNs.

Both networks seem to be impacted by similar time-steps in the input. The LSTM has the theoretical ability to remember further back, yet it seems to choose not to. The simple features are already inherently incorporating previous information through the moving averages, perhaps this information is sufficient for our networks and a longer term memory is unnecessary. Another explanation is vanishing gradients. During backpropagation, the gradient from a single point is propagated throughout all time steps in the sequence which may cause it to either explode or vanish. LSTMs can constrain this effect through the use of the forget gate but it is still not entirely guaranteed to avoid this complication. Another aspect of the figures is that the gradients from the LSTM appear more smooth and consistent with regards to the sign of the gradient. The gradients of the simple RNN exhibit on the other hand a more chaotic pattern with gradients of a higher magnitude but with less certainty in how to utilize them. The difference between the architectures boils down to the forget gate. Due to the non-stationarity of financial data, the patterns a RNN finds may quickly become irrelevant. Removing the RNNs ability to forget leads to a higher uncertainty in how each feature and timestep affects the output. The LSTM on the other hand can itself choose when to reset its memory state and thus has a more updated perspective on the current properties of the time-series. Our hypothesis is that when contradictory patterns from the past can be excluded from the current predictions, the LSTM has a clearer view of which features affect the output positively or negatively. Hence we see a consistent smooth slope in the gradient.
We can make the same investigation with respect to the Sharpe Ratio. Figure

5.20 reveals the gradients with respect to our complex features.

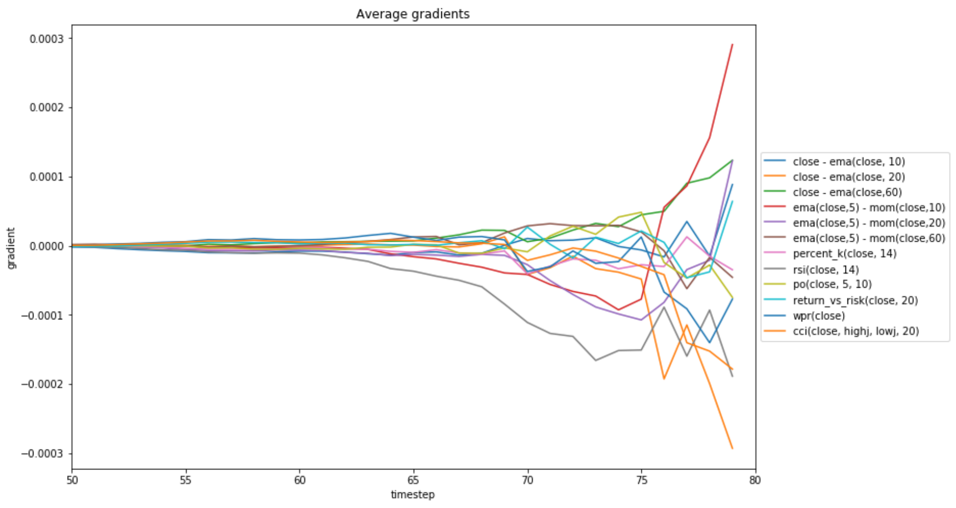


Figure 5.20: The average impact of our complex features for the LSTM.

Comparing Figure 5.20 with Figure 5.18 we note that models trained with the complex features exhibit a longer memory compared to models utilizing the simple features. This is inline with the discussion in the previous section where we noted that the complex features are a collection of quicker and more revealing features. The short lookback periods on the features force the networks to tailor features which incorporate a longer time-period of observations. The features may also reveal more complex patterns in the market than the simple features. To fully utilize these patterns, a combination of a larger set of past observations is needed. Perhaps moving averages filter out important aspects the time series, making it difficult for the LSTM to see the value in employing a larger memory to make predictions. It is interesting to also consider the quickest feature of them all, they lag-1 differenced closing price.

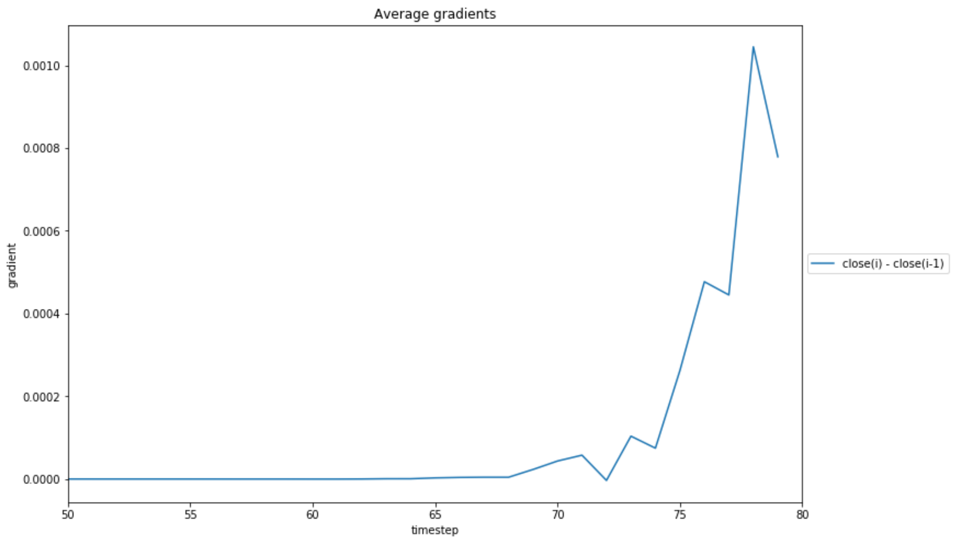
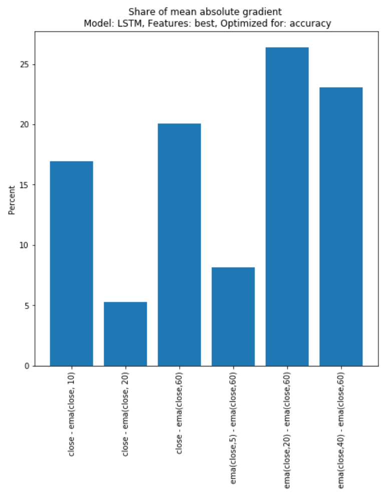


Figure 5.21: The gradients with respect to the differenced closing price.

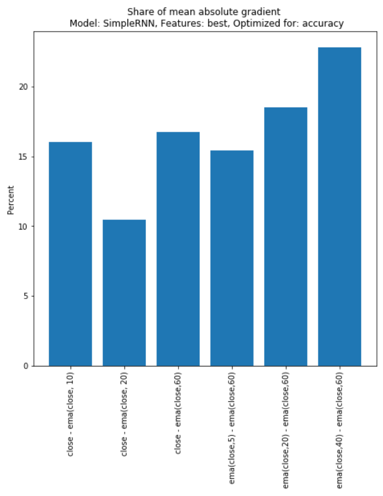
Figure 5.21 reveals the gradient of our single feature when the output is the next days directional movement. The lookback period of our model seems to be similar to that of the simple features. Given only the next days prediction, it is not entirely misguided to assume that the most relevant features are only the most recent. The value of the gradients increases as we approach the last timestep. This is akin to the network creating an exponential moving average where more weight is placed on the more recent observations.
It is difficult to provide a clear cut answer if our results stem from difficulties in model optimization or if they reveal a deeper fact about the Markovian nature of the market. It is however clear that there is a difference between how the input data affects the predictions of the LSTM and simple RNN respectively. This further highlights the importance of the forget gate in Recurrent architectures.

5.4.2 Feature importance

Next we will take a closer look at the individual features and how they impact the decisions different models. Consider the figures from the previous section. We determine absolute feature importance by considering the absolute values of the gradients and computing the mean value throughout the time series. Thus we measure the average impact that each feature has on the last data-point. Figure 5.23 depicts the mean absolute gradient of a number of recurrent networks optimized for accuracy using the simple features. The values have been rescaled for the purpose of clarity to depict the percentage share of the total gradients.



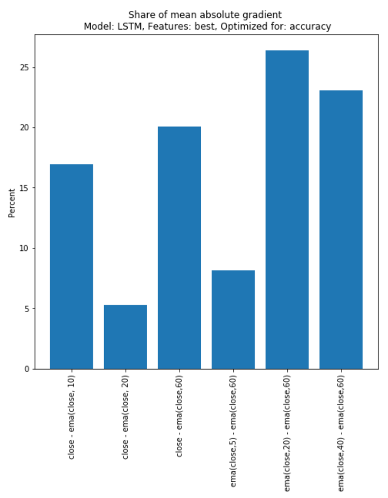
(a)



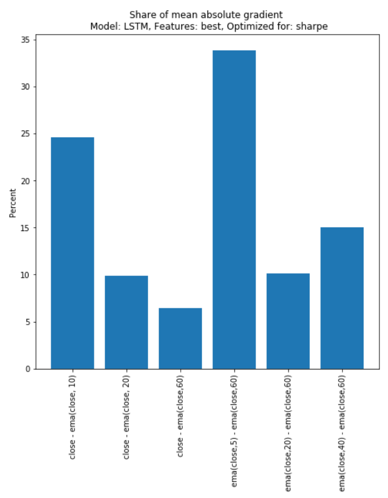
(b)

Figure 5.22: Mean percentage share of the gradients of each feature of LSTM networks optimized for accuracy. The mean gradients are found by computing the gradient of the last data point with respect to each input in the series and averaging the absolute value.

In figure 5.22*b* we see a more equal distribution of gradients. The indicates that the network has a level of stability with respect to the input. If we one day encounter an outlier in some feature, the networks performance may not be significantly altered, since the weight of the gradients is distributed equally among the features. Equally distributed feature significance could either indicate that each feature is equally important, or similarly equally insignificant. From our earlier results on evaluating the accuracy on our networks, the simple recurrent network gives the least promising results. Thus the features importance in the case of the simple RNN could indicate that it has not found any significant patterns determined by a subset of the features. Figure 5.22*a* gives a more unequal distribution of gradients for each feature. This gives us further insight into how our model makes its decisions and whats features it considers important. Leaving the simple RNN behind, we proceed by providing a comparison between features considered by the LSTM when optimizing for Sharpe and optimizing for Accuracy.



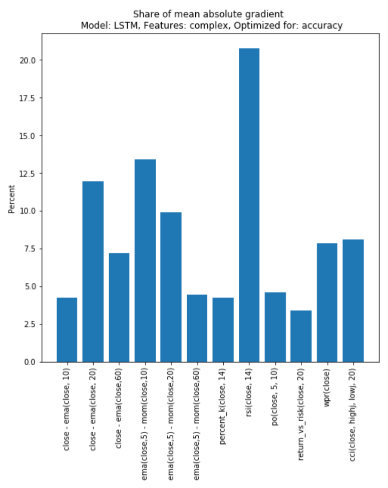
(a) Feature importance for an LSTM optimized for accuracy.



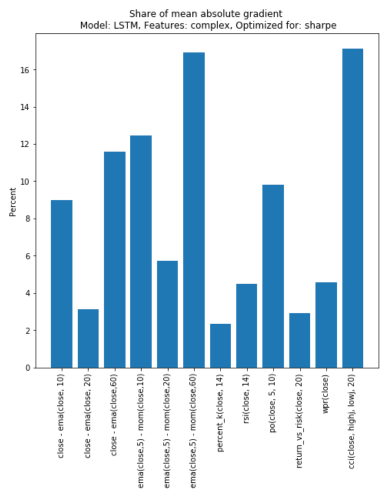
(b) Feature importance of the simple features for an LSTM optimized for Sharpe.

Figure 5.23: Comparison between LSTM optimized for accuracy and sharpe. In the case of sharpe, the output seems to be determined for the most part by the first and third feature.

The LSTM optimized for sharpe weighs a smaller subset of features higher than those optimized for accuracy. The features belonging to the simple features are all very similar since the are all different combinations of moving averages. One network converging to a point where it considers feature n the most important could be more related to parameters in the network optimization rather than the feature having an intricate connection to market predictability. The complex features provide a more diverse set of features which provides us with more insight than the simple ones.



(a) LSTM optimized for accuracy.



(b) LSTM optimized for sharpe

Figure 5.24: Comparison between LSTM optimized for accuracy and sharpe on the complex features.

We can begin by considering Figure 5.24*a*. There seems to be significant weight placed on the moving average features on the left hand side of the diagram. This is consistent with the fact that simple features predict accuracy the best. The largest proportion of the gradients stem from the Relative Strength Index which indicates if an asset is overbought or oversold. Perhaps including this feature among the simple features would give us further improvement in our main results. Next we consider the results for the models optimized for Sharpe. The two most significant features is the 5-day exponential moving average minus the momentum and the Commodity Channel Index. The latter feature is specifically interesting since it is a measure of the number of standard deviations the current price is from its moving average. When an asset is volatile, it strays far away, measured in standard deviations, from the current mean. Thus the CCI gives a measure of future risk in the market and could this be used for adjusting our positions when optimizing for Sharpe. Besides moving averages, another mid-significant feature is the Percentage Price Oscillator.

Before we draw any further conclusions it is best to question the validity of this approach. The gradients do show how features impact the models predictions. The question is whether we can completely disregard features with "low" significance or if they work conjointly with other features in a way that is not exposed with this method. If we had a linear model, the gradient would expose the coefficient of each factor and would thus be a direct measure of importance given that each feature is scaled similarly. In our non-linear model we cannot so readily draw the same conclusions. There is significant difference between model performance when evaluating on the test set. The absolute performance itself may however be too insignificant to provide an accurate framework for investigating the model further. Neural networks are non-identifiable models. This implies that given two models with the same performance, it is not necessary that the model parameters are the same. Thus we can have models with similar, good performance yet evaluating the gradients would yield different results. This is due to the fact that different random initializations of the variables and optimizers during training yield different local minimas.

Chapter 6

Discussion

In this section we will further discuss the results presented in the previous chapter and discuss the limitations of our research and possible extensions.

6.1 Discussion of results

6.1.1 Feed-forward Networks

The results show clearly that simple features are to be preferred to the complex. Layer size and residual connections seem to also have a significant impact on the performance. Utilizing an ensemble of models can greatly boost performance, specifically if the average individual performance of the models is good. The results are however very unstable which sheds light on the issue of using Neural Networks for this kind of task. Furthermore a larger hyperparameter space could have been explored. Different optimizers with varying learning rates together with varying the batch size could be explored. A comparison with the best hyperparameters chosen for each layer size was conducted but this result could be strengthened by further testing various setups.

Batch size is intricately connected to the evaluation of the Sharpe ratio since the batch-size is used as the time period we calculate the Sharpe ratio on. We hypothesize that a larger batch size allows the model to create positions which take into account larger trends and thus possibly allowing longer holding periods for each asset. A small batch-size would provide the model with insufficient information to be able to take positions in an intelligent way. In this thesis a batch-size of 256 trading days is used which corresponds to about a year. Investigating further batch-sizes may prove a valuable insight into understanding how the network optimizes for maximum Sharpe.

6.1.2 Recurrent Networks

The recurrent networks were investigated in a similar way to the feed-forward networks. The layer depth and sizes were also chosen similarly. It is worth mentioning that a LSTM has four times the model parameters of a feed-forward network given the same number of units. This is due to the fact that a LSTM unit utilizes four different transformations of its inputs. Thus it is further motivated to investigate additional layer sizes than the ones chosen. The input size of our series is also a parameter that could be explored. As our gradients show, only the past 10 inputs play a part in our predictions. Transforming the input data to take this into account may provide us with better results. It could also be worth investigating a rolling window approach, where a window of k days is used to predict the movement of $(k+1)^{st}$ day. In this method we do not propagate the state of the LSTM throughout the network and thus force it to forget after k days. This could provide further improvements specifically for the simple RNN.

6.1.3 Multi-loss Networks

The multi-loss networks were highly dependent on the weights chosen for the dual loss functions. The loss weights are in turn highly dependent on the magnitude of the loss functions. For an accuracy of 50%, a binary cross-entropy loss lies at a value of $ln(2) \approx 0.69$. The negative Sharpe loss however proved to lie in the range of -10 to -5 during training. When combing these losses, it becomes more valuable for the network to decrease the Sharpe loss rather than increasing the accuracy. This is apparent when evaluating the accuracy of a multi-loss network. Introducing a Sharpe loss to the model quickly deteriorated our results, even when placing much greater importance on the binary cross-entropy loss. The other side of this coin is that when evaluating Sharpe, introducing accuracy layers proved successful since the network still placed greater importance on minimizing Sharpe loss and we achieved our desired regularizing effect. One can further this discussion by considering the fact that it is the gradients of the loss function that influences how the parameters are updated, not just the magnitude of the loss. A detailed investigation of how to scale the magnitude of the losses w.r.t to the magnitude of the gradients could be conducted.

6.1.4 Gradients

How and why neural networks work is a field still in its infancy. Various methods have been proposed to measure the relationship between input and output such as measuring the mutual information between them [1]. Gradient based methods have also been explored on image recognition [16] to discover what parts of the image is the most important when the network performs classification. Similar to what an image classifier finds interesting in an image, gradient based methods performed on RNNs reveal what the most interesting parts of sequence are. The difficulty of the analysis performed in this thesis stems from the fact that the performance of our RNNs may not be significant enough to be able to excerpt any useful properties from the gradients. In our analysis, the gradients are averaged among a set of the best performing models. If a gradient is significant in all of them, it will still be significant in the mean. This provides a grain of legitimacy for our method but still leaves room for vast improvement. An obvious extension is introducing more models, to see if they all find similar features important. A factor that is ignored in our analysis is the impact different markets have on each other. Perhaps exploring the effect the features from different currency pairs or different commodities with similar properties affect each other.

6.2 Discussion of scope and limitations

This thesis aims to answer three propositions:

1. Can a Recurrent neural network outperform a Feedforward network given the same feature setup?
2. What is the influence of the objective function of the Neural Network and how does it impact feature selection?
3. What set of features are most significant to the task of predicting future returns?

6.2.1 Can a Recurrent neural network outperform a Feed-forward network given the same feature setup?

The foundation for a fair comparison between feed-forward networks and RNNs lies in the use of the same input data for the model and evaluating on the same performance metric. A third setup that must be taken into account is hyperparameter search. Through trial and error, the network setups that were chosen in this thesis were considered because they at an initial stage provided seemingly the best performance. A hyperparameter search was then conducted on these networks to further fine tune performance. The field of Neural Networks is relatively new and theoretical results regarding optimal network architecture are few. Instead, state-of-the-art results are found through intuition and an extensive exploration of the hyperparameter space. We can never state that our results are final, since we never reach a global minimum of our loss function. Yet the combination of results and intuition tells a story that is coherent, both with existing research and properties of the financial market. This provides a grain of evidence for the validity of our results and opens up research questions which could further help confirm them.

6.2.2 What is the influence of the objective function of the Neural Network and how does it impact feature selection?

The impact of the objective function was examined both through the lens of single loss networks and multi-loss networks. In the first comparison the two different performance metrics were evaluated on both networks. The comparison is however not entirely fair. When evaluating the Sharpe of a model optimized for accuracy, the predictions are scaled to integers in the set $\{-1, 1\}$. Positions are thus deemed to be absolute, either we buy or sell. When optimizing for Sharpe, the outputs are continuous variables in the range of $[-1, 1]$. This gives the model the freedom to adjust positions based on its certainty in predictions. It is not surprising that a unrestricted model can achieve a higher Sharpe ratio. However this is not entirely relevant for our study. Our comparison still shows significant positive results for all combinations of loss functions and metrics. The relationship is hypothesized to be evidence for a deeper connection between the objective functions. This relationship is then further investigated through the use of multi-loss networks.

The issue regarding our multi-loss configuration was previously addressed in our discussion. When optimizing for Sharpe and incorporating an auxiliary loss function, we achieve an improved score. Here we cannot draw the conclusion whether this is simply a regularizing effect or if the networks share important features. A study combining additional loss functions could be conducted which could shed further light on this issue.

6.2.3 What set of features are most significant to the task of predicting future returns?

Our simple and complex features are continuously compared throughout all our experiments where the results paint a coherent picture that simple features are to prefer when optimizing for accuracy and complex features are better suited for the task of optimizing for the Sharpe ratio. Explanations are provided where we highlight the different properties of the features and provide an intuition as to why certain features may be preferred for different tasks and networks. We provide an additional comparison between features in our analysis of the model gradients. Here our reasoning is limited in part by the assumption that the most significant features for an asset will stem from those generated from the same assets time-series. Further limitations are apparent when questioning the stability of our models and thus the significance of our gradients. There is no clear cut method to perform feature selection. As with many methods in Machine Learning, the best practice is to test every combination. We present an alternative method to perform the feature selection though further research is needed to confirm the validity of the approach.

6.3 Future work

We have previously touched upon specific further research that could possibly improve our analysis. In this section we will expand upon these areas and present a set of questions which could provide a basis for further research within recurrent neural networks and financial modeling.

6.3.1 Feature importance and selection

Our gradient based method is one of many methods to examine the inner workings of our models. Information theoretic approaches are becoming increasingly popular and provide an interesting framework for further investigating the connection between features and model predictions. Tishby et al laid the foundation for an information theoretic approach in their paper "Deep Learning and the Information Bottleneck Principle" [17]. Here the mutual information between input and output is examined. Another recent interesting paper is "Learning to Explain: An Information-Theoretic Perspective on Model Interpretation" by Chen et. al. [1] which utilizes information theory to specifically investigate feature selection.

6.3.2 Convolutional Neural Networks

Equipped with our new-found understanding of Recurrent Networks, one conclusion we can draw is that perhaps not the entire past sequence of data is required to perform accurate predictions. This opens up to models where our input can be a sequence of fixed length. Convolutional Networks work through a set of kernels which measures the dependency between neighboring inputs. Time series analysis is thus a natural application of these types of models since neighboring data points have a clear time-dependent relationship. Past research has shown that convolutional neural networks outperform LSTMs on these types of tasks, both in machine translation [7] and on financial modeling [12].

6.3.3 Transfer Learning

A popular area of Machine Learning today is Transfer Learning and specifically Domain Adaption [5]. Given two datasets with similar properties, one can train a classifier on one dataset and extend it to function on the other set as well. An application on financial modeling would be to train a model using inputs from one asset class and then extending it to work on a different market. The process would shed light on relationships between different markets and could further increase model performance similar to how our multi-loss networks worked. Another possibility is investigating whether patterns that appear on an intraday level could also be used on daily data. Data collected on a minute-to-minute basis provides a several magnitudes larger dataset than the daily closing prices. We could thus train a better fitted model on the intraday set and extend it to work on daily data.

Chapter 7

Conclusion

From our results it is clear that recurrent networks outperform feed-forward networks on the task of predicting both price movements and maximizing Sharpe on financial time series. The difference in performance is recurring in both simple single loss networks and multi-loss networks. It shows that utilizing the time dimension in addition to the features give a significant advantage in predicting future returns.

The difference between our two feature setups is also continually evaluated throughout our results. For predicting directional accuracy, the simple features outperform the complex setup. The opposite holds when maximizing the Sharpe ratio. The feature setups reveal different aspects of the market. Different objective functions allow us to explore different ways in how the networks utilize the features. A conclusion we can draw is that the complex features reveal more information about market risk, which allows the network to perform a better optimization compared to using the simple features. Introducing a regularization through multi-loss networks also gives a larger performance. This can be due to the fact that our objective functions share similar goals which would imply that they would benefit from sharing some parameters.

The recurrent networks gradients are analyzed. From looking at the memory of RNNs, it is concluded that a large lookback period may by unnecessary for our recurrent setups. The gradients show that for well-trained networks, there are features which exhibit a higher influence over the networks decisions. Here differences in feature importance between different optimization goals further highlight the fact that the feature setups reveal different aspects of the market.

Bibliography

[1] Jianbo Chen, Le Song, Martin J. Wainwright, and Michael I. Jordan. Learning to explain: An information-theoretic perspective on model interpretation. *CoRR*, abs/1802.07814, 2018.

[2] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[3] François Chollet et al. Keras. `https://keras.io`, 2015.

[4] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[5] Nicolas Courty, Rémi Flamary, Devis Tuia, and Alain Rakotomamonjy. Optimal transport for domain adaptation, 2015.

[6] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks, 2015.

[7] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. *CoRR*, abs/1705.03122, 2017.

[8] Magnus Hansson. On stock return prediction with lstm networks. 2017.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[10] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[11] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[12] Zhengyao Jiang, Dixing Xu, and Jinjun Liang. A deep reinforcement learning framework for the financial portfolio management problem, 2017.

[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[14] Andrew W Lo. The adaptive markets hypothesis: Market efficiency from an evolutionary perspective. 2004.

[15] Scott Patterson. *The quants: how a small band of math wizards took over Wall St. and nearly destroyed it.* Crown, 2010.

[16] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps, 2013.

[17] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. *CoRR*, abs/1503.02406, 2015.

[18] Philip Widegren. Deep learning-based forecasting of financial assets. 2017.

[19] Lingxue Zhu and Nikolay Laptev. Deep and confident prediction for time series at uber. 2017.

TRITA -SCI-GRU 2018:262