DEGREE PROJECT IN MATHEMATICS,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2021

Value at Risk Estimation with Neural Networks

A Recurrent Mixture Density Approach

WILLIAM KARLSSON LILLE

DANIEL SAPHIR

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ENGINEERING SCIENCES

Abstract

In response to financial crises and opaque practices, governmental entities and financial regulatory bodies have implemented several pieces of legislature and directives meant to protect investors and increase transparency. Such regulations often impose strict liquidity requirements and robust estimations of the risk borne by a financial firm at any given time. *Value at Risk* (VaR) measures how much an investment can stand to lose with a certain probability over a specified period of time and is ubiquitous in its use by institutional investors and banks alike. In practice, VaR estimations are often computed from simulations of historical data or parameterized distributions.

Inspired by the recent success of Arimond et al. (2020) in using a neural network for VaR estimation, we apply a combination of recurrent neural networks and a mixture density output layer for generating mixture density distributions of future portfolio returns from which VaR estimations are made. As in Arimond et al., we suppose the existence of two regimes stylized as bull and bear markets and employ Monte Carlo simulation to generate predictions of future returns. Rather than use a swappable architecture for the parameters in the mixture density distribution, we here let all parameters be generated endogenously in the neural network. The model's success is then validated through Christoffersen tests and by comparing it to the benchmark VaR estimation models, i.e., the mean-variance approach and historical simulation.

We conclude that recurrent mixture density networks show limited promise for the task of predicting effective VaR estimates if used as is, due to the model consistently overestimating the true portfolio loss. However, for practical use, encouraging results were achieved when manually shifting the predictions based on an average of the overestimation observed in the validation set. Several theories are presented as to why overestimation occurs, while no definitive conclusion could be drawn. As neural networks serve as black box models, their use for conforming to regulatory requirements is thus deemed questionable, likewise the assumption that financial data carries an inherent pattern with potential to be accurately approximated. Still, reactivity in the VaR estimations by the neural network is significantly more pronounced than in the benchmark models, motivating continued experimentation with machine learning methods for risk management purposes. Future research is encouraged to identify the source of overestimation and explore different machine learning techniques to attain more accurate VaR predictions.

Keywords: *Machine learning, Neural networks, LSTM, MDN, Mixture density, Value at Risk, VaR, Risk, Financial mathematics, Finance*

Value at risk estimering med neurala nätverk
En recurrent mixture density approach

Sammanfattning

I respons till finanskriser och svårfattlig verksamhetsutövning har överstatliga organ och finansmyndigheter implementerat lagstiftning och utfärdat direktiv i syfte att skydda investerare och öka transparens. Sådana regleringar förelägger ofta strikta likviditetskrav och krav på redogörelse av den finansiella risk som en marknadsaktör har vid en given tidpunkt. *Value at Risk* (VaR) mäter hur mycket en investering kan förlora med en viss sannolikhet över en på förhand bestämd tidsperiod och är allestädes närvarande i dess användning av institutionella investerare såväl som banker. I praktiken beräknas estimeringar av VaR framför allt via simulering av historisk data eller en parametrisering av densamma.

Inspirerade av Arimond et als (2020) framgång i användning av neurala nätverk för VaR estimering applicerar vi en kombination av "recurrent" neurala nätverk och ett "mixture density output"-lager i syfte att generera mixture density-fördelningar för framtida portföljavkastning. Likt Arimond et al. förutsätter vi existensen av två regimer stiliserade som "bull" och "bear" marknader och applicerar Monte Carlo simulering för att generera prediktioner av framtida avkastning. Snarare än att använda en utbytbar arkitektur för parametrarna i mixture density-fördelningen låter vi samtliga parametrar genereras endogent i det neurala nätverket. Modellens framgång valideras via Christoffersens tester samt jämförelse med de prevalenta metoderna för att estimera VaR, det vill säga mean-variance-metoden och historisk simulering.

Vår slutsats är att recurrent mixture density-nätverk enskilt uppvisar begränsad tillämpbarhet för uppgiften av att uppskatta effektiva VaR estimeringar, eftersom modellen konsekvent överestimerar den sanna portföljförlusten. För praktisk användning visade modellen däremot uppmuntrande resultat när dess prediktioner manuellt växlades ner baserat på ett genomsnitt av överestimeringen observerad i valideringsdatat. Flera teorier presenteras kring varför överestimeringen sker men ingen definitiv slutsats kunde dras. Eftersom neurala nätverksmodeller agerar som svarta lådor är deras potential till att bemöta regulatoriska krav tveksam, likväl antagandet att finansiell data har ett inneboende mönster kapabelt till att approximeras. Med detta sagt uppvisar neurala nätverkets VaR estimeringar betydligt mer reaktivitet än i de prevalenta modellerna, varför fortsatt experimentation med maskininlärningsmetoder för riskhantering ändå kan vara motiverat. Framtida forskning uppmuntras för att identifera källan till överestimeringen, samt utforskningen av andra maskininlärningsmetoder för att erhålla mer precisa VaR prediktioner.

Nyckelord: *Maskininlärning, Neurala nätverk, LSTM, MDN, Mixture Density, Value at Risk, VaR, Risk, Finansiell matematik, Finans*

Acknowledgements

We would like to express our most sincere gratitude to our supervisor at KTH, Alessandro Mastrototaro, for his excellent guidance, attention to detail, and useful insights throughout the project. Additionally, we thank Jimmy Olsson for orchestrating the supervision in unison with Mastrototaro.

To our external project partner Scila AB, we direct a huge thanks for letting us collaborate with you to write this thesis, and especially Björn Thornquist who has been an ideal partner in terms of discussing scope, results, and their implications for practical use within the financial industry.

Special thanks to Rasmus Guterstam and Vidar Trojenborg for providing formidable company and encouragement throughout this thesis.

William Karlsson Lille & Daniel Saphir
Stockholm, May 2021

Contents

1 Introduction

Mathematical finance has in recent years become increasingly dependent on applying statistical machine learning techniques to inform or support decision making [KTH]. Despite the capricious nature of financial data, there have been important advancements in making accurate financial forecasts using advanced machine learning. While the use of machine learning in finance is not new in and of itself, technological advancements have extended the realm of possibilities for financial modeling. Concurrently, in the wake of financial crises and in response to opaque business practices, the subject of financial risk and transparency has begun to consume a greater part of the discussion. Several pieces of legislature have been formulated since the 2008 financial crisis aiming to increase transparency and protect investors, such as MIFID-II and Basel-III [NSS]. Many of these legal frameworks require financial firms to continually calculate different measurements of risk and diligently report these to the appropriate supervisory authority (such as Finansinspektionen in Sweden). Aside from regulatory purposes, accurate risk assessments are of course beneficial for the firm in question in order to make more well-informed decisions when in- or divesting in different securities.

One of the most common measures of risk is *Value at Risk* (VaR) [HH]. In essence, VaR is the amount of money one stands to lose from an investment over a specified time period and with a specified confidence level. If the 1-week 5 % VaR of an investment is \$100 it means that the investment will with 95 % certainty not result in a loss exceeding \$100. Equivalently, the loss will exceed \$ 100 with a probability of 5 %. The most common methods for calculating VaR are usually divided into the parametric and non-parametric approaches. For the former, a normal distribution is often assumed wherein one constructs a normal distribution with parameters μ and σ^2 being the mean and variance of the investment based off a sample of previous returns; this is referred to as the *mean-variance* approach. For the non-parametric case, the most common method is *historical simulation*, in which the VaR is chosen as an empirical quantile of historical returns. Both of these methods have their respective pros and cons, yet a common advantage is their relative simplicity and low computational requirements. A common disadvantage is that both methods fail to adequately capture the tail risk of an investment, i.e., the worst possible outcomes.

In 1994, Christopher Bishop formulated the idea of *mixture density networks* (MDNs), in which the output layer of a neural network produces parameters for a mixture of distributions which should in principle be able to "represent arbitrary conditional probability distributions in the same way that a conventional neural network can represent arbitrary functions" [BC], which gives credence to the notion of more accurately capturing the tail risk of an investment. While documentation on the use of MDNs exists in several domains, their application in a financial context is more sparsely documented. Inspired by the recent success of Arimond et al. [ABHKW], this thesis wishes to explore the use of mixture density networks for VaR prediction, with the distinction of letting all parameters be produced within the network rather than only the mixture probabilities. To evaluate the performance of the MDN approach, the mean-variance and historical simulation methods are used as benchmarks. Improvement upon the benchmark methods would entail increased accuracy in VaR prediction and imply significant financial benefits by enabling investors to take on greater risk and improve liquidity.

1.1 External Project Partner

At the intersection of advanced technology and financial risk lies the Swedish fintech company Scila, who provides risk mitigation services in a standardized way. This enables their customers to conform to the plethora of regulations imposed by regulatory institutions, without deviating from their core business. Scila is independent of trading venues and focuses solely on product development. The purpose of this thesis is to explore ways in which Scila can improve their risk management services by utilizing state-of-the-art machine learning techniques benchmarked against established methods. This project is thus conducted under the supervision and for the benefit of Scila AB. Björn Thornquist, head of product development, will act as supervisor on Scila's behalf. In analogue to the Swedish copyright act (1960:729) article 40 a §, the authors of this thesis relinquish any and all intellectual property developed within the framework of this thesis to Scila AB.

1.2 Previous Research

Bishop introduced the idea of combining conventional neural networks and mixture density models in 1994, demonstrating their capability on robot inverse kinematics. "Long Short-Term Memory" (LSTM) networks were introduced in 1997 by Hochreiter and Schmidhuber, a form of recurrent neural network well suited for time series data capable of learning patterns between lags of arbitrary sizes [H&S]. In 2013, Alex Graves successfully utilized LSTMs with a MDN-output layer for generating sequences of realistic handwriting [AG]. Inspired by the success of Gu, Kelly, and Xiu in applying neural networks for measuring asset risk premia [GKX], Arimond et al explored the usage of several network architectures and a MDN-layer for estimating VaR-thresholds [ABHKW]. In this context, the mixture probabilities (of which there were two) are stylized as regime probabilities, i.e., probabilities of being in a bull or bear market - markets where prices are generally on the rise or decline, respectively. In their paper, only the mixture probabilities are produced within the network and established models (such as Hidden Markov Models) are applied to infer the distribution parameters. Thus, there exists some incentive for a purely neural network based approach to either promote or eschew the idea of using MDNs for estimating VaR. Additionally, documentation on the usage of MDNs in a financial context is scarce, including a robust outline of the underlying methodology.

1.3 Objective

The objective of this thesis is to explore the potential of LSTM-MDNs in the task of generating VaR predictions. Specifically, a neural network with LSTM-layers and a mixture density output layer is to be used for constructing a mixture density distribution to model future returns of a portfolio. Moreover, due to the probabilistic nature of mixture densities, a secondary objective of this thesis is to explore the network's ability to detect bull and bear regimes within a financial market.

1.4 Scope and Limitations

As the literature regarding the usage of LSTM-MDNs in a financial context is quite scarce, this thesis aims to explore its potential while providing deeper insights on the methodology as well as the generated results. Moreover, the data used has been manually extracted from open sources such as *Google Finance*. As such, there are some limitations in regard to the quality and availability of old financial data. Due to this, 20 years of historical data was used in order to ensure quality and

availability. In relation to this, considering the aim of this thesis, only adjusted[1] historical stock price data from one financial market was used (see 3.2.1). The idea is that the findings of this thesis can later be extended to evaluate LSTM-MDNs on different asset types and markets.

1.5 Research Questions

This thesis aims to answer the following research questions:

- To what extent can LSTM-MDNs produce more accurate sampling distributions for VaR estimation than prevailing methods (i.e., historical simulation, mean-variance approach)?
- Can LSTM-MDNs successfully detect market regimes based on historical data?
- If successful, are LSTM-MDNs economically viable as a VaR model regarding model training, computation time, data requirements, and interpretability?

[1] Closing price amended to reflect the value of the stock after taking corporate actions (such as dividend payouts and splits) into account.

2 Theoretical Framework

2.1 Financial Theory

The following section is intended to give a rudimentary basis for the financial themes of this thesis, primarily regarding the prerequisite knowledge necessary to apply the subsequent statistical methods.

2.1.1 Financial Forecasting

A seminal problem in finance is forecasting future asset prices. According to the *Efficient Market Hypothesis* (EMH), the asset price contains all available information about the asset [EF]. As such, by the EMH, the asset is priced correctly, and profits cannot be generated on the basis of the asset being under- or overvalued. Despite the ubiquity of the EMH in academic texts, both individual and institutional investors seek to outperform the market in order to generate positive returns on their portfolios. This thesis will focus on quantitative approaches for such an objective, which to some extent necessitate making numerically driven guesses for future returns. Success in this domain has recently been achieved by Gu, Kelly and Xiu in their paper *Empirical Asset Pricing via Machine Learning* in which artificial neural networks are employed to measure risk premia [GKX]. The authors attribute the success of neural networks in this context to their capability of capturing non-linear predictor interactions among the data. Whereas other prediction methods necessarily require some formulation of the function and relationships among variables (such as e.g. a multiple linear regression), neural networks have been shown to be *universal approximators*, i.e. they can determine the relationship between input and output given that some such mapping truly exists [HSW]. The conjecture in this thesis is thus that there indeed does exist a mapping from previous returns to future returns, and that it can be learned by a neural network. However, rather than predicting a point forecast, by using a mixture density network one instead acquires a range of potential returns as conditioned by a distribution. This is significantly less strict than assuming a one-to-one function, while still maximizing the vast potential neural networks have to offer.

2.1.2 Financial Data

When working with financial data, there are a number of beneficial adjustments one could and in many cases should make to the data before modeling. First, let the current time be denoted by 0 and consider a time in the future denoted by T. We consider also a portfolio at time T to be some function of the vector $\mathbf{S}_T$ of n stock prices, i.e. $\mathbf{S}_T = [S_T^1, S_T^2, ..., S_T^n]$ where the superscript $i = 1, 2, ..., n$ denotes the i:th asset. It is further assumed that one has access to samples of the share price vector from the D equally spaced points in time preceding the time T in the future, including the current time i.e. $[\mathbf{S}_{-D}, \mathbf{S}_{-D+1}, ..., \mathbf{S}_0]^{\mathrm{T}}$; negative subscripts here represent times in the past, and the superscript T denotes transposition, thus giving us a $(D+1) \times n$-matrix containing prices of the n assets over the last D equally spaced points in time as well as the current prices. For this example we assume that these are daily stock prices, but in general any equidistant time unit may be used, e.g. weeks, months, quarters, milliseconds etc. The future time T is thus effectively one day (or other time unit) in the future: $T = 1$. The stock prices may be quite strongly dependent on each other on a day-to-day basis, while prices far apart in time may be very different. For any

given day $t \neq -D$, the vector of stock prices $\mathbf{S}_t = [S_t^1, S_t^2, ..., S_t^n]$ may be transformed into a vector of stock returns $\mathbf{R}_t = [R_t^1, R_t^2, ..., R_t^n]$, where each R_{-t}^i is defined as

$$R_{-t}^i = \frac{S_{-t}^i}{S_{-t-1}^i} - 1 \tag{1}$$

for $t = 0, ..., D-1$, as there is no access to the price of stock i before day -D. Rather than a matrix of daily stock prices, we now have a $D \times n$-matrix daily returns: $[\mathbf{R}_{-D+1}, \mathbf{R}_{-D+2}, ..., \mathbf{R}_0]^T$. This transformation is done due to the assumption that returns are more weakly dependent than prices, and nearly identically distributed [HH]. This is illustrated in the figures below: the upper panel shows the adjusted closing price of Alphabet Inc. plotted for each day from 2004-08-19 to 2020-12-31, whereas the corresponding returns are shown in the lower panel. It is clear that price changes when viewed in nominal terms are highly dependent (the price of an asset on day t is a good indication of what the price will be on day $t+1$), while this is significantly harder to say for adjacent returns.

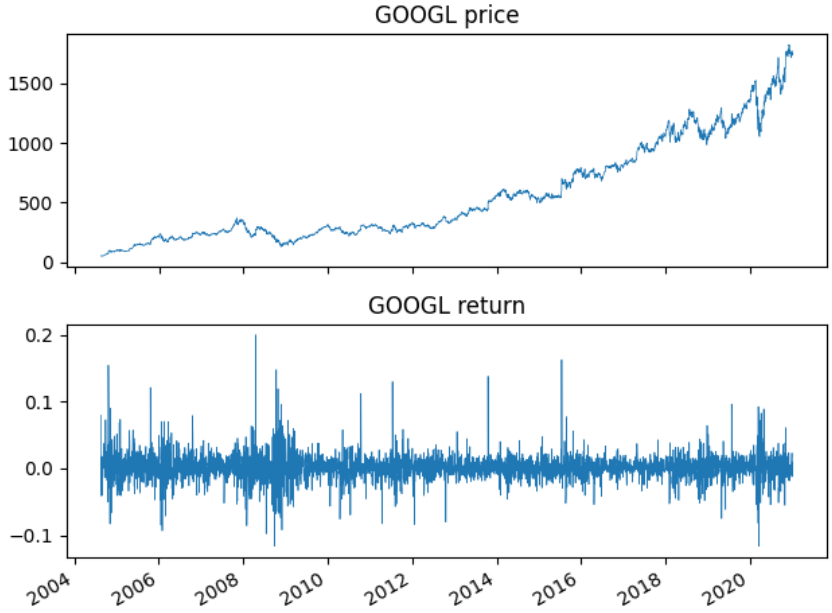

Figure 1: Alphabet's (ticker: GOOGL) adjusted closing price from 2004 to 2020 (above) and Alphabet's daily return from 2004 to 2021 (below).

Moreover, the distributional characteristics of past returns are to some extent representative for $\mathbf{R}_T$, the vector of percentage returns for the future time T which one desires to forecast. The abstraction is effectively that the returns serve as nearly independent and identically distributed (i.i.d.) observations of random variables drawn from the unknown probability distribution of V_T. We denote the portfolio return by V_T, and it is given by some function f of the portfolio assets' returns, i.e. $V_T = f(\mathbf{R}_T)$, on which statistical methods can be applied to determine the distribution. Considering that this approach assumes historical samples of returns to be samples from a probability distribution of a *future* portfolio value, there is an implicit assumption that past changes are

indicative of future changes. Whether or not this is true is a subject of great debate, yet imperative if one wishes to use statistical methods for forecasting of financial data. For this thesis, returns are thus used and the portfolio function f is discussed in section 3.2.2.

2.1.3 Risk

A fundamental aspect in finance is the trade-off between risk and reward. Vast research has been made in optimizing the trade-off in such a way that the maximum profit is attained while exposure to risk is minimized. The most seminal paper on this subject was penned by Harry Markowitz in 1952, whose eponymous portfolio seeks to diversify over an asset universe in order to maximize the risk-adjusted return [HM]. A measure of risk-adjusted return is the Sharpe ratio:

$$\text{Sharpe Ratio} = \frac{R_p - R_f}{\sigma_p} \tag{2}$$

Where R_p is the return on the portfolio, R_f the risk-free rate and σ_p the standard deviation of the portfolio's excess return. In the Markowitz framework, risk is thus operationalized as variance. While this constructional choice is intuitive and simple, it lacks the depth necessary for real-world application. To illustrate this, consider two portfolios with equal expected return and variance; these would have the same Sharpe ratio, yet the form of the variance might look extremely different. One portfolio could have a certain variance due to small, frequent losses while the other could have the very same variance owing to rare but very massive losses. While a perfectly rational investor (*Homo Economicus* [2]) would assign these portfolios equal value, most investors would not want exposure to massive losses, regardless of the potential upside. This phenomenon is called *loss aversion* and its existence was formulated by prominent behavioral economists Kahneman and Tversky in 1984 [K&T]. In fact, Kahneman and Tversky postulated that inviduals are twice as averse to losses as they are prone to gains; in other words, for an investment with a potential loss of \$1, the potential gain must be \$2 or higher. Accepting this as true, one can understand the need for more robust measures of risk than variance, which in fact does not distinguish between positive and negative returns. Since the work of Sharpe and Markowitz, a multitude of more useful risk measures have been developed; *Value-at-Risk* or *VaR* will be the focal point of this thesis and is defined in section 2.2.1.

2.1.4 Market Regimes

Market trends are often stylized as *bear* and *bull markets*. In bear markets, prices tend to fall and investors exhibit pessimism toward future price development, whereas the opposite holds true for bull markets. While the exact criteria for being in a bull or bear market varies in the literature, this thesis will assume that states characterized by lower or even negative returns and more volatility are bear markets while those with higher or even positive returns are bull markets. As such, the period during and after the financial crisis of 2008 was famously bearish, while the following years up until 2020 have undoubtedly been more bullish. While these regimes have been clearly noticeable from simply regarding index charts, this thesis seeks to identify more short-term regime changes, which are significantly more difficult to notice by the naked eye. It is often easy to identify the current

[2]The term Homo economicus or "Economic man" was coined by John Stuart Mill and later popularized by behavioral economists such as Richard Thaler and Daniel Kahneman. The term signifies a hypothetical market actor with infinte capacity to make rational decisions.

overarching market regime, but it would be beneficial for investors to identify the current market sentiment automatically and accommodate one's investments accordingly.

2.2 Mathematical Theory

The following section outlines the necessary financial mathematics necessary to understand the thesis' main task, i.e., generating precise VaR predictions.

2.2.1 Value at Risk

One of the most widely used measures of risk is *Value-at-Risk* (VaR). Formally, the VaR at level α of the profit and loss distribution X is defined as

$$VaR_\alpha(X) = \min\{m : P(m \cdot r_f + X < 0) \leq \alpha\} \tag{3}$$

where r_f is the risk-free rate [HH]. In turn, the right hand side of (3) can be rewritten as

$$\begin{aligned} &\{m : P(m \cdot r_f + X < 0) \leq \alpha\} \\ &= \{m : P(-X/r_f > m) \leq \alpha\} \\ &= \{m : 1 - P(-X/r_f \leq m) \leq \alpha\} \\ &= \{m : P(-X/r_f \leq m) \geq 1 - \alpha\} \end{aligned} \tag{4}$$

Setting $L = -X/r_f$ and defining X as $X = V_1 - V_0 r_f$, where V_t is the portfolio value at time t, one can rewrite L as

$$L = -X/r_f = -(V_1 - V_0 r_f)/r_f = V_0 - V_1 r_f \tag{5}$$

Thus, equation 5 shows that L is the discounted portfolio loss. An alternative, yet equivalent, formulation of $VaR_\alpha(X)$ in equation 3 is thus

$$VaR_\alpha(X) = \min\{m : P(L \leq m) \geq 1 - \alpha\} \tag{6}$$

Using 6, one can thus interpret $VaR_\alpha(X)$ as the smallest m such that the probability of the loss being at most m is at least $1 - \alpha$. In other words, $VaR_\alpha(X)$ estimates how much an investment X could potentially lose with a given probability α over a certain time period. For example, if the one-week 5 % VaR of an investment is \$100, there is a 5 % probability that the investment will lose \$100 or more by the next week.

Moreover, $VaR_\alpha(X)$ is the $1 - \alpha$-quantile of L. The q-quantile of the random variable L with distribution function F_L is by definition

$$F_L^{-1}(q) = \min\{m : F_L(m) \geq q\} \tag{7}$$

With the notation in (7), we see from (6) that $VaR_\alpha(X)$ thus can be written as

$$VaR_\alpha(X) = {F_L}^{-1}(1 - \alpha) \tag{8}$$

In order to estimate VaR, there is a need to determine the distribution function F_L, in accordance with 7 [HH]. In essence, there are two types of approaches to doing so: parametric and nonparametric approaches. The most prominent methods for doing so are the mean-variance approach

(parametric) or by historical simulation (non-parametric), and as such these methods will be used as benchmarks against the thesis' developed model.

Mean-variance method: In parametric VaR methods, one uses a statistical model to fit a distribution with estimated parameters to describe a future return R_T. A widely used assumption for financial data is that this distribution, conditioned on past data (observations of R_t for $t < T$), tends to be a Gaussian distribution with some mean μ and variance σ. Using the obtained distribution, one can then simply compute the $1-\alpha$-quantile and estimate $VaR_\alpha(X)$ as that value. The method's main advantage is its simplicity and speed, as fitting a Gaussian distribution to data is a readily available feature accessible in a multitude of software packages.

Rather than use the whole historical time window up to the current time $t = 0$, a so-called lookback period is chosen, here denoted d, from which the mean and standard deviation is calculated. The choice of lookback period is in the hands of the analysts, however using the whole period for longer time horizons is not advised as different periods are likely to have a large discrepancy in means and volatilities. Using this method entails that the VaR is not calculated for the first d days. Thereafter, for each d-day period, one calculates

$$\hat{\mu}_t = \frac{1}{d}\sum_{j=0}^{d-1} R_{t-d+j}, \text{ for } t = 1, ..., T \tag{9}$$

The variance is calculated as:

$$\hat{\sigma}_t^2 = \frac{\sum_{j=0}^{d-1}(R_{t-d+j} - \hat{\mu}_t)^2}{d-1}, \text{ for } t = 1, ..., T \tag{10}$$

The standard deviation is of course the square root of the above expression, $\hat{\sigma}_t$, also referred to as the volatility. Effectively, equations 9 and 10 are moving averages. For a day t in $[1, T]$, the VaR prediction at confidence level α is then generated by taking the $1-\alpha$ quantile of the distribution $\mathcal{N}(\hat{\mu}_t, \hat{\sigma}_t)$.

Historical Simulation: In principle, historical simulation works in the same way as the mean-variance approach, i.e., by choosing a lookback period and calculating the 95 %-quantile; however, in historical simulation, this is done by simply ordering the observed returns and choosing the empirical quantile rather than parametrizing and then choosing the quantile. If given access to a sample of independent copies $L_1, ..., L_n$ of the loss distribution L for an investment X, the empirical estimate of $\mathrm{VaR}_\alpha(X)$ is given by

$$\widehat{\mathrm{VaR}}_\alpha(X) = L_{\lfloor n\alpha \rfloor + 1, n} \tag{11}$$

where $L_{1,n} \geq \ldots \geq L_{n,n}$ is the sample of losses ordered ascendingly by size, and $\lfloor \cdot \rfloor$ denotes the floor function[3]. To elucidate this with an example, suppose that there are $n = 101$ samples of the loss L and one seeks the VaR at confidence level $\alpha = 0.05$. This would require ordering the samples such that $L_{1,101} \geq \ldots \geq L_{101,101}$ and choosing $\widehat{\mathrm{VaR}}_{0.05}$ as $L_{\lfloor 101 \cdot 0.05 \rfloor + 1, 101} = L_{\lfloor 5.05 \rfloor + 1, 101} = L_{6,101}$.

[3]The floor function of x is defined as: $\lfloor x \rfloor = \max\{m \in \mathcal{Z} \mid x \leq m\}$

2.2.2 Mixture Density Distributions for VaR

Implicitly, one thus assumes that all past changes in risk factors follow the same distribution and that future values will do so as well, both in the historical and mean-variance case. Due to the vast amount of potentially nonlinear interactions and extreme (often unforeseeable) events prevalent in financial data, relying on a single parameterized distribution or historical simulation may not adequately capture the tail risk of an investment. Thus, there is an incentive for a modeling framework which samples from different distributions, to account for varying market conditions. The idea of this is to increase the precision of the VaR estimate by letting the distributions vary across historical samples as they might be subdued or enhanced by different macroeconomic or time-dependent factors. One can interpret e.g. two regimes as bull and bear markets, with corresponding mixture probabilities ($\hat{\pi}_k,\ k = \{1, 2\}$) conditioning the frequency in which a distribution (e.g. Gaussian with parameters $\hat{\mu}_k$ and $\hat{\sigma}_k$) is sampled from. Bishop postulated that arbitrary conditional probability distributions could be approximated by mixture density networks, which inspires the route taken in this thesis. Of dubious certainty is the assumption that return distributions are static and thus appropriately modeled by mixture densities. As such, a mixture density enables modelling an overall population wherein there exists K sub-populations with different probability distributions. A mixture density can thus be defined as

$$P(y|x) = \sum_{k=1}^{K} \pi_k(x)\phi_k(y|x) \text{ with } \sum_{k=1}^{K} \pi_k = 1 \tag{12}$$

where ϕ_k is the associated probability density for each subpopulation. For the example of bull and bear markets described above, assuming ϕ_k to be Gaussian for each k, ϕ_k is thus defined as

$$\phi_k(y|x) = \mathcal{N}(y\,|\mu_k(x), \sigma_k^2(x)) = \frac{1}{\sigma_k(x)\sqrt{2\pi}} \exp\left(-\frac{||(y - \mu_k(x))||^2}{2\sigma_k^2(x)}\right) \tag{13}$$

Since there is no theoretical quantile available for mixture density distributions, the procedure for acquiring VaR estimations boils down to Monte Carlo simulation. The procedure is as follows:

1. Generate predictions of mixture density distribution parameters $(\hat{\pi}_k, \hat{\mu}_k, \hat{\sigma}_k)$ for each day in $[0, T]$
2. Using the outputs from step 1, randomly select a regime as conditioned by a Bernoulli distribution; regime 1 is thus chosen with a probability of $\hat{\pi}_1$ and regime 2 with a probability of $\hat{\pi}_2 = 1 - \hat{\pi}_1$. Draw a sample from the corresponding distribution $\mathcal{N}(\hat{\mu}_k, \hat{\sigma}_k)$
3. Repeat step 2 a large amount of times for each day
4. Order the simulated losses by size and choose the empirical quantile at the desired confidence level.

The theory underlying mixture densities in neural networks is detailed in section 2.3.5.

2.2.3 Backtesting VaR-Models

An important aspect in constructing a VaR model is validating its results. A classic approach to verifying the legitimacy of the model, as suggested by Kupiec, is simply to check whether the

number of VaR estimations smaller than the true return of an asset falls within the confidence level [KP]. That is, for a confidence level of $\alpha = 5\%$, the number of such violations should be approximately 5% of the total data points, i.e. αN. If this holds true, the model is thus considered accepted.

This type of violation testing lies as a base for multiple other validation methods. Before continuing, we must first define a violation sequence. We let the true loss of sample n be denoted by L_n and the estimated loss at confidence level α or $\widehat{\text{VaR}}_\alpha$ of sample n be denoted by $\hat{L}_{n,\alpha}$. Furthermore, we let $I_n(A)$ denote the indicator function of sample n, i.e.

$$I_n(A) = \begin{cases} 1 \text{ if the event A occurs,} \\ 0 \text{ else} \end{cases} \tag{14}$$

We may now define a violation sequence from $n = 1$ to N as

$$I_{n=1}^{N}(\alpha) = \{I_1(\hat{L}_{1,\alpha} < L_1),\ I_2(\hat{L}_{2,\alpha} < L_2), ...,\ I_N(\hat{L}_{N,\alpha} < L_N)\} \tag{15}$$

$I_{n=1}^{N}(\alpha)$ is thus a list of ones and zeros denoting whether a breach has occurred or not over all samples N , respectively.

A test suggested by Christoffersen uses the principle of VaR violations but focuses rather on the correlations between them [CP]. As such, Christoffersen shows that VaR validity corresponds to confirming the following two hypotheses [CSD]:

- The unconditional coverage hypothesis: the probability of the true return being smaller than the estimated VaR must be equal to the confidence level. That is, for a violation sequence $I_{n=1}^{N}(\alpha)$

$$P(I_n(\alpha) = 1) = E[I_n(\alpha)] = \alpha \tag{16}$$

- The independence hypothesis: VaR violations occurring at different times must be independently distributed. That is, $I_n(\alpha)$ is independent from $I_{n-k}(\alpha), \forall k \neq 0$ meaning that VaR violations in the past do not hold information on future or current violations. Under the independence hypothesis, the violations follow a binomial distribution

$$\sum_{n=1}^{N} I_n(\alpha) \sim Bin(N, \alpha) \tag{17}$$

To evaluate the two hypotheses, Christoffersen suggests the conditional coverage (CC) test, which takes the independence (IND) hypothesis and the unconditional coverage (UC) hypothesis into account. With this, Christoffersen suggests that $I_{n=1}^{N}(\alpha)$ is modeled by a Markov Chain with two states (violation and no violation) and transition matrix

$$\Pi = \begin{pmatrix} \pi_{00} & \pi_{01} \\ \pi_{10} & \pi_{11} \end{pmatrix} \tag{18}$$

where $\pi_{ij} = P(I_n(\alpha) = j \,|I_{n-1}(\alpha) = i)$. By modelling the violations as such, for some confidence level α, the probability of a violation in the current period depends on the existence of a violation in the previous period. The null hypothesis is thus formulated as follows:

$$H_0 : \Pi = \Pi_\alpha = \begin{pmatrix} 1-\alpha & \alpha \\ 1-\alpha & \alpha \end{pmatrix} \tag{19}$$

Moreover, due to its construction, the probability of a violation at time t is independent from the state in t. We denote the maximum likelihood estimator of Π under the alternative hypothesis (i.e. that $\Pi \neq \Pi_\alpha$) as

$$\widehat{\Pi} = \begin{pmatrix} \hat{\pi}_{00} & \hat{\pi}_{01} \\ \hat{\pi}_{10} & \hat{\pi}_{11} \end{pmatrix} \tag{20}$$

and the number of violations in the violation sequence as

$$V = \sum_{n=1}^{N} I_n(\hat{L}_{n,\alpha} < L_n). \tag{21}$$

Christoffersen then shows that under H_0 the likelihood ratio statistic LR is defined as

$$LR = -2 \ln \left[\frac{(1-\alpha)^{N-V} \cdot \alpha^V}{(1-\hat{\pi}_{01})^{n_{00}} \hat{\pi}_{01}^{n_{01}} \cdot (1-\hat{\pi}_{11})^{n_{10}} \hat{\pi}_{11}^{n_{11}}} \right] \tag{22}$$

where n_{ij} is the number of occurrences when $I_t(\alpha) = j$ and $I_{t-1}(\alpha) = i$, is asymptotically $\chi^2(2)$-distributed, allowing for acceptance of the null hypothesis if the generated p-value is ≥ 0.05 [HT].

2.3 Neural Networks

In general, neural networks work similar to a brain with neurons that are activated, deactivated, and connected to one another with synapses. Neural networks are constructed using an input layer, hidden layer(s), and an output layer. Each layer consists of neurons that communicate with one another in order to transport information from the input layer to the output layer, letting the information pass through the neurons in the hidden layers, in which they are transformed according to predetermined functions. What information that should be passed from a neuron in one layer to a neuron in the next layer is decided by the network's vector of weights w, which thus determine the strength of the influence between the interconnected neurons. The main idea of neural networks is to function as a "black-box" model which can identify nonlinear relationships between variables.

We first consider a standard "vanilla" neural network, i.e., a fully connected feed-forward neural network with one hidden layer containing M_1 neurons. The input of observations x_i for $i = 1, 2, ..., n$ is transformed into inputs a_j for $i = 1, 2, ..., M_1$ which are then fed to the hidden layer in which an activation function σ is applied resulting in the outputs y_k for $k \geq 1$. We begin by defining a_j more precisely, letting the superscript (1) represent the hidden layer.

$$a_j^{(1)} = \sum_{i=1}^{n} w_{ji}^{(1)} x_i + w_{j,0}^{(1)} \tag{23}$$

where $w_{j,0}$ is a bias weight influencing a_j. This j:th input a_j, with j spanning over all nodes in the given layer, is transformed using an activation function σ to get Z_j; since the activation function may be different between layers, we suppress its index, but note that the same activation function is applied to all units within the same layer.

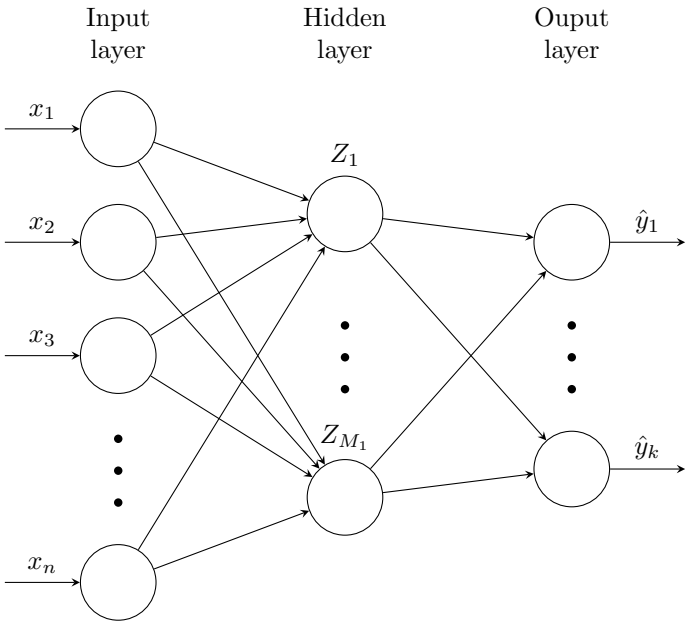

Figure 2: A neural network with one hidden layer.

$$Z_j^{(1)} = \sigma(a_j^{(1)}) = \sigma\left(\sum_{i=1}^{n} w_{ji}^{(1)} x_i + w_{j,0}^{(1)}\right) \tag{24}$$

These are the hidden units in the neural network. In turn, the procedure of producing the k:th output $\hat{y}_k$ from the hidden units is done in the output layer (2)

$$\hat{y}_k = \sigma\left(\sum_{j=1}^{M_1} w_{kj}^{(2)} Z_j^{(1)} + w_{k,0}^{(2)}\right) = \sigma\left(\sum_{j=1}^{M_1} w_{kj}^{(2)} \sigma\left(\sum_{i=1}^{n} w_{ji}^{(1)} x_i + w_{j,0}^{(1)}\right) + w_{k,0}^{(2)}\right) \tag{25}$$

With $M_1, M_2, M_3..., M_L$ representing the number of neurons in layer M_l. As mentioned before, this is for the case with one hidden layer and its architecture is shown in figure 2 below.

Expanding the network's architecture to multiple hidden layers, i.e., a deep neural network, the transition from layer $m-1$ to layer m is then given by

$$Z_j^{(m)} = \sigma\left(\sum_{i=1}^{M_{(m-1)}} w_{ji}^{(m)} Z_i^{(m-1)} + w_{j,0}^{m}\right) \tag{26}$$

with m being the current hidden layer.

Let $\mathcal{L}(w)$ denote some convex loss function that includes the parameters of the network, e.g., mean squared error defined as

$$\mathcal{L}(w) = \frac{1}{2}\sum_{n=1}^{N}(\hat{y}_n - y_n)^2 \tag{27}$$

With $\hat{y}_n$ being defined by 25 and N being the total number of inputs x, equation 27 thus contains all parameters of the network. In order to optimize the network, the aim is to find w^* such that $\nabla\mathcal{L}(w^*) = 0$. This is done by choosing a starting weight w^0 and iteratively updating it in order to minimize the loss function. The updates for the τ:th iteration are defined as

$$\begin{aligned} w^{(\tau+1)} &= w^{(\tau)} + \Delta w^{(\tau)} \\ \text{with } \Delta w^{(\tau)} &= -\zeta\nabla\mathcal{L}(w^{(\tau)}) \end{aligned} \tag{28}$$

Where $\zeta > 0$ is defined as the learning rate of the network. This process is called gradient descent. This methodology is quite slow as it involves evaluating $\nabla\mathcal{L}(w)$ as defined in equation 27 for all $n = 1, .., N$. Nevertheless, the process of updating the weights w is done in two steps; evaluating $\nabla\mathcal{L}(w)$ and updating the weights according to $\nabla\mathcal{L}(w)$. In order to evaluate this, the chain rule allows for redefining $\nabla\mathcal{L}(w)$ as follows

$$\begin{aligned} \frac{\partial\mathcal{L}_n}{\partial w_{ji}^{(m)}} &= \frac{\partial\mathcal{L}_n}{\partial a_j^{(m)}}\frac{\partial a_j^{(m)}}{\partial w_{ji}^{(m)}} \\ \text{where } \frac{\partial a_j^{(m)}}{\partial w_{ji}^{(m)}} &= Z_j^{(m-1)} \\ \rightarrow \frac{\partial\mathcal{L}_n}{\partial a_j^{(m)}}Z_j^{(m-1)} &=: \delta_j Z_j^{(m-1)} \end{aligned} \tag{29}$$

where the earlier expressions for input a_j and hidden units Z_j have been used. This implies that it is enough to calculate $\delta_j = \partial\mathcal{L}_n/\partial a_j^{(m)}$ in order to evaluate $\nabla\mathcal{L}_n(w)$. Starting at the output of a neural network with L layers, the output $\hat{y}_{nj}$ can be thought of conceptually as the input to the $(L+1)$:th layer, $a_j^{(L+1)}$. Using the loss function $\mathcal{L}(w)$ as defined in equation 27, one thus has

$$\begin{aligned} \frac{\partial L_k}{\partial a_j^{(L+1)}} &= \frac{\partial}{\partial a_j^{L+1}}\left(\frac{1}{2}(\hat{y}_{nj} - y_{nj})^2\right) \\ &= \frac{\partial}{\partial a_j^{L+1}}\left(\frac{1}{2}(a_j^{(L+1)} - y_{nj})^2\right) \\ &= a_j^{L+1} - y_{nj} \\ &= \hat{y}_{nj} - y_{nj} \end{aligned} \tag{30}$$

For computation of δ_j for units inside the network, the chain rule applies as well

$$\delta_j^{(m)} = \frac{\partial\mathcal{L}_n}{\partial a_j^{(m)}} = \sum_{s=1}^{M_{m+1}}\frac{\partial\mathcal{L}_n}{\partial a_s^{(m+1)}}\frac{\partial a_s^{(m+1)}}{\partial a_j^{(m)}} \tag{31}$$

Through equation 31 and by $Z_j^{(m)} = \sigma(a_j^{(m)})$, the following expression for $\delta_j^{(m)}$ is found

$$\begin{aligned}
\frac{\partial a_s^{(m+1)}}{\partial a_j^{(m)}} &= \frac{\partial}{\partial a_j^{(m)}} \sum_{j=1}^{M_m} w_{sj}^{(m+1)} \sigma(a_j^{(m)}) = \sigma'(a_j^{(m)}) w_{sj}^{(m+1)} \\
\frac{\partial \mathcal{L}_n}{\partial a_s^{(m+1)}} &= \delta_s^{(m+1)} \\
\delta_j^{(m)} &= \sigma'(a_j^{(m)}) \sum_{s=1}^{M_{m+1}} w_{sj}^{(m+1)} \delta_s^{(m+1)}
\end{aligned} \tag{32}$$

The weights are then updated in accordance with equation 28 and 29 until the loss is as small as possible. This process is called the back-propagation algorithm.

2.3.1 Activation Functions

Activation functions are used to teach neural networks about different patterns in the data. The functions used determine what information is relayed from a neuron in one layer to a neuron in the subsequent layer. The necessity of activation functions is plural and includes limiting the signals between neurons to certain values and perhaps most importantly to capture nonlinear patterns in the data. While there exist several different activation functions useful in different situations, the following are featured in this thesis and will be detailed below.

Sigmoid: The sigmoid activation function as implemented in most machine learning packages is defined as

$$S(x) = \frac{1}{1 + e^{-x}} \tag{33}$$

There are actually different types of sigmoid functions, which all share the properties of being monotonic and having a bell-shaped first derivative. The function defined above is known as the *logistic function*. The logistic function converges to 1 when $x \to \infty$ and 0 when $x \to -\infty$ and is thus often used to represent probabilities.

ReLU: Rectified linear unit or *ReLU* is an activation function defined as

$$R(x) = \max(0, x) \tag{34}$$

Thus, the function only returns that which is positive. It has as of late become one of the most ubiquitous and powerful activation functions in deep neural networks [LRZ].

Tanh The hyperbolic tangent function or *tanh* is defined as

$$T(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{35}$$

Tanh is also a sigmoid function, and is in fact simply a shifted and scaled version of the logistic function defined in (33), an important difference being that *tanh* returns values between $[-1, 1]$.

Softmax The *softmax* activation function is defined as

$$sm(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}, \text{ for } i = 1, ..., K \text{ and } \mathbf{x} = x_1, ..., x_K \in \mathbb{R}^K \tag{36}$$

The *softmax* function thus results in the input being mapped to a set of values each between 0 and 1, much like the logistic function, however here all components sum to 1 jointly rather than each being a value which can be interpreted as a probability. The *softmax* function is thus appropriate for the task of creating mixture density weights, since the mixture probabilities should sum to unity.

ELU The exponential linear unit (ELU) activation function is defined by the following equations:

$$E(x) = \begin{cases} x, & x > 0 \\ e^x - 1, & x \leq 0 \end{cases} \tag{37}$$

This function thus returns values > -1, and for very negative values quickly approaches zero. If one seeks to model e.g. portfolio variances, the function thus yields the desired behavior of being nonnegative if the function is shifted upward by 1, while simultaneously not increasing exponentially if the input is itself positive. In the figure below, the ELU activation function has thus been increased by 1 to ensure positive values.

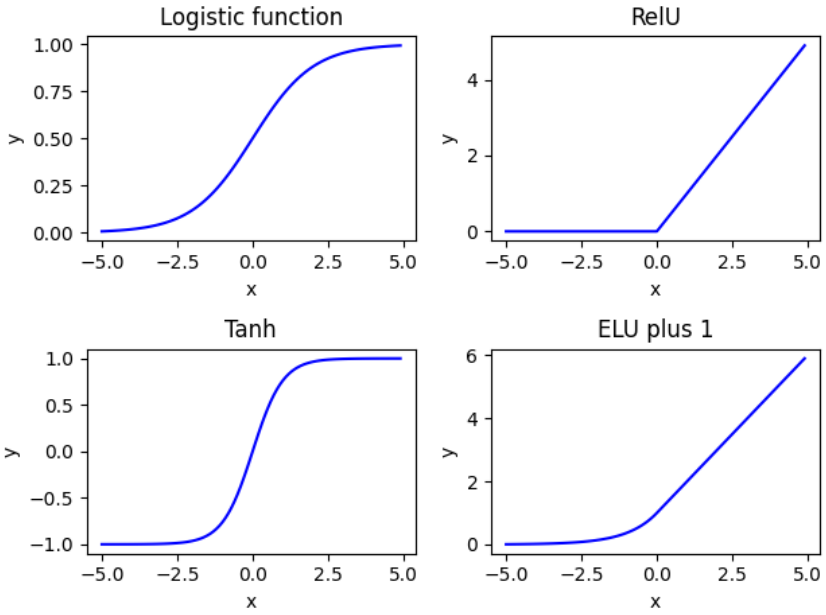

Figure 3: Different activation functions and their values for $x \in [-5, 5]$.

2.3.2 Optimizers

Traditional gradient descent is only one among a plethora of methods available for updating the weights in a neural network. The method is advantageous in the sense that it is simple to implement

and will return the global optimum if the objective function is convex. One important caveat is that the method might converge to a *local* optimum and hence cease searching for the global optimum. Moreover, for each iteration the computational complexity will be high, and for large data sets potentially infeasible. This is due to the fact that all training data is used in each iteration; letting the number of samples be denoted by N and the dimension of the data set be D, the computational complexity will for each iteration be $\mathcal{O}(ND)$. The technique of *Stochastic Gradient Descent* was developed to mitigate this issue, as well as the risk of getting stuck in a local minima [SCZZ].

Stochastic Gradient Descent: Here, one sample of a small batch of samples is randomly chosen to update the gradient during each iteration, as opposed to computing the exact value of the gradient using the entire data set. The new weights are thus calculated as in equation 28, the difference being that this is only done for one sample i rather than for the entire data set. In doing so, the calculation cost is reduced significantly; since the number of samples here is equal to one the computational complexity will be $\mathcal{O}(D)$. The direction of the gradient descent does not uniformly converge in one direction as the normal gradient descent method due to only one sample being used since this introduces additional noise. Furthermore, having one learning rate for all features (where some might be very prevalent and others rarely so) is problematic, since ideally rarely occurring features should have larger gradient updates and vice versa. One method that addresses this issue is the "adaptive gradient algorithm", or *Adagrad*.

Adagrad: Adagrad aims to adapt the learning rate to the problem's parameters by assigning a low learning rate for frequently appearing parameters and a high learning rate for rarely appearing ones. The update equations are in the Adagrad case:

$$\begin{aligned} \nabla\mathcal{L}(w^{(\tau+1)}) &= \frac{\partial\mathcal{L}(w^{(\tau)})}{\partial w}, \\ V^{(\tau)} &= \sqrt{\sum_{i=1}^{\tau}\nabla\mathcal{L}(w_i)^2 + \epsilon}, \\ w^{(\tau+1)} &= w^{(\tau)} + \Delta w^{(\tau)} \\ \text{with } \Delta w^{(\tau)} &= -\zeta\frac{\nabla\mathcal{L}(w^{(\tau)})}{V^{\tau}} \end{aligned} \tag{38}$$

Here, $V^{(\tau)}$ is the accumulate historical gradient of parameter w at τ, w^{τ} is the value of w on iteration τ, and ε is set to a very small value to counteract division by zero. Consequently, the learning rate is not held constant and is updated by investigating the historical gradients preceding the current iteration. One problem is that the learning rate will converge toward zero if the training time is long. The methods *RMSProp* and *AdaDelta* were developed to mitigate this issue. In these methods, rather than accumulating *all* historical gradients, one instead investigates the gradients over a shorter period and uses an exponential moving average to calculate the second-order cumulative momentum like so:

$$V^{(\tau)} = \sqrt{\beta V^{(\tau-1)} + (1-\beta)(\nabla\mathcal{L}(w^{(\tau)})^2} \tag{39}$$

Where β is some exponential decay parameter. The flexible learning rates from AdaGrad and momentum-based methods like AdaDelta and RMSProp are combined in the relatively new (2015)

optimizer "adaptive moment estimation", or *Adam*.

Adam: Adam stores exponentially decaying averages of past gradients $g^{(\tau)}$

$$\begin{aligned} g^{(\tau)} &= \beta_1 g^{(\tau-1)} + (1-\beta_1)\nabla\mathcal{L}(w^{(\tau)}), \\ V^{(\tau)} &= \sqrt{\beta_2 V^{(\tau-1)} + (1-\beta_2)(g^{(\tau)})^2} \end{aligned} \tag{40}$$

With β_1 and β_2 being some exponential decay rates, resulting in the weight update equations

$$\begin{aligned} w^{(\tau+1)} &= w^{(\tau)} + \Delta w^{\tau} \\ \text{with } \Delta w^{(\tau)} &= g^{(\tau)} - \zeta \frac{\sqrt{1-\beta_2}}{1-\beta_1} \frac{g^{(\tau)}}{V^{(\tau)} + \epsilon} \end{aligned} \tag{41}$$

where ϵ is set to a very small value to ensure that there is no division by 0. In combining these two approaches, one is left with an optimizer which is computationally efficient and simultaneously minimizes the need for prior learning rate tuning [K&L].

2.3.3 Batch Size and Epochs

The neural network does its training by splitting the input data into batches and letting this number of samples propagate through the network. For a given training data set containing N samples and a batch size denoted b, the network divides N into N/b batches of size b. For each batch, weight updates are made and its result is shown in the value of the network's loss function. When all batches have been propagated through the network, the training has been performed over 1 epoch. That is, the number of epochs is the number of times that all batches are propagated through the network. Naturally, there are implications to choosing the batch size as it affects the gradient descent explained above. Typically, the choice of batch size introduces three types of gradient descent: batch gradient descent (where $b = N$), mini-batch gradient descent (where $1 < b < N$) and stochastic gradient descent (where $b = 1$). The figure below illustrates how each approach behaves when trying to reach a global optimum for the gradient.

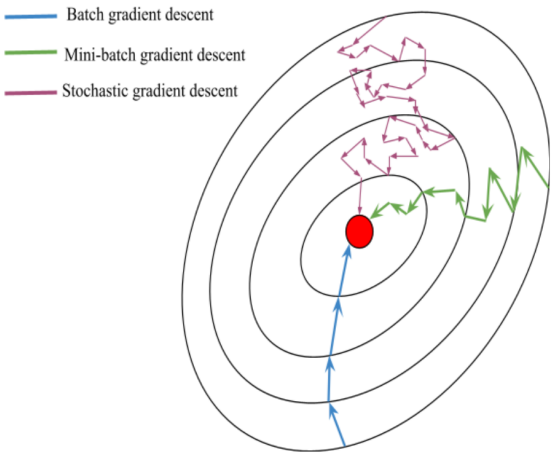

Figure 4: Illustration of gradient descent using different batch sizes.

While having a smaller batch size might be beneficial in terms of computational time since the weights are updated after each propagation, a smaller batch size might make the value of the gradient less accurate. The figure below illustrates how the gradient fluctuates more for the stochastic approach compared to the batch approach. As such, a mini batch gradient descent approach serves as a trade-off between the two.

2.3.4 Long Short-Term Memory Network

There are a plethora of different types of neural networks, each being more suitable depending on the task at hand. For time series, the recurrent neural network (RNN) is commonly used as it includes a temporal component and is thus able to capture dynamic and time-dependent behaviors in the data. A certain type of RNN is the Long Short Term Memory-Neural Network (LSTM-NN), which, based on configuration of the model, can consolidate information from far in the past with that which is more recent. A LSTM-NN includes at least one LSTM-cell which is constructed as shown in figure 5.

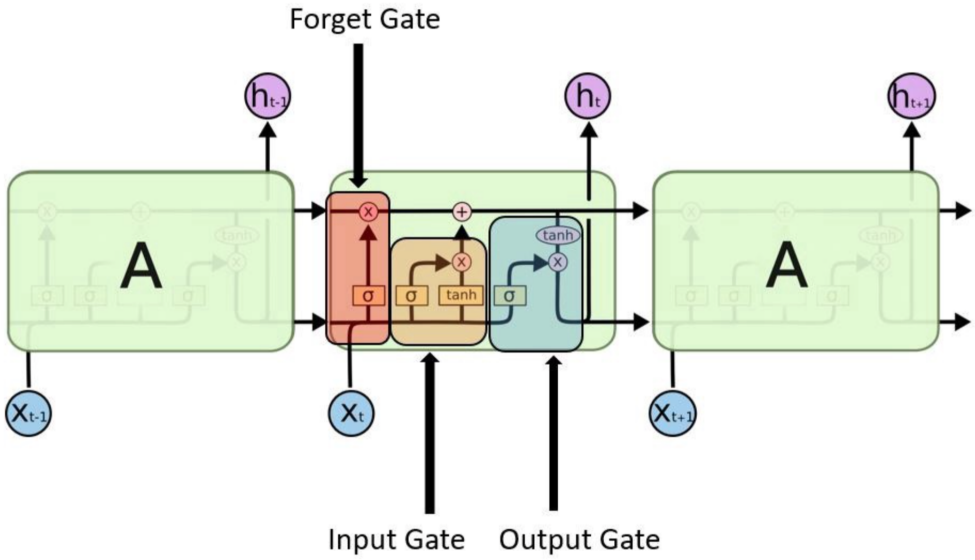

Figure 5: Illustration of LSTM-cell architecture (image used with permission from creator Christopher Olah [CO].

The LSTM-cell consists of a forget gate, an input gate, and an output gate. The main purpose of the LSTM-cell is to remember information over time intervals while the gates control the information that flows in and out of the cell. Essentially, the gates compute which information that is to be remembered and which information that is to be forgotten. The information that is remembered in the LSTM-cell is stored in its cell state C_t and is calculated in accordance with the input x_t, the computation in the forget gate f_t as well as the previous output h_{t-1}.

In the forget gate, the previous output h_{t-1} is concatenated with the new input x_t. Using the

sigmoid activation function, σ, the output of this gate is between 0 and 1 and can be defined as follows

$$f_t = \sigma(W_f Z_t + b_f) \tag{42}$$

Where $Z_t = [h_{t-1}, x_t]$, i.e. the concatenation of previous output and current input, W_f is a weight matrix and b_f is a bias vector. The value of f_t is then point-wise multiplied with the previous cell-state C_{t-1}. The aim is to reduce the elements of C_{t-1} whose corresponding elements in f_t are close to zero, i.e., the cell state is updated and current state elements that carry lower weights are reduced. Simply put, these elements are "forgotten" and its effect on the LSTM output is therefore reduced.

The input gate works similarly to the forget gate using Z_t, but produces the output e_t. As can be seen in figure 5, the Z_t is fed both through a sigmoid activation as well as a *tanh* activation. The produced output of the *tanh* function are the candidate values, c_t. Both e_t and c_t hold a corresponding weight matrix W_e, W_c and a bias vector b_e, b_c.

$$\begin{aligned} e_t &= \sigma(W_e Z_t + b_e) \\ c_t &= \tanh(W_c Z_t + b_c) \end{aligned} \tag{43}$$

The two outputs e_t and c_t are then point-wise multiplied similarly as in the forget gate to reduce elements that are to be ignored, or deemed not important enough, in c_t. c_t thus contains the information that is to be updated in C_{t-1} in order to obtain the current cell state C_t hence the name candidate values. This is done through adding the output of the forget gate with the output of the input gate. The full update to the current cell state can thus be portrayed as follows

$$C_t = e_t \cdot c_t + f_t \cdot C_{t-1} \tag{44}$$

The output gate produces the output u_t which essentially is used to filter how the memory is influencing the output of the cell. Similar to the forget and input gates, u_t has its corresponding weight matrix W_u and bias vector b_u. Since the idea of the output gate is to reduce and enhance the importance of the generated vectors, a sigmoid function is used to get a scalable value between 0 and 1.

$$u_t = \sigma(W_u Z_t + b_u) \tag{45}$$

u_t is then point wise multiplied with the current cell state C_t in equation 44 albeit scaled by a *tanh* activation function to control the values generated by the addition in equation 44. This multiplication thus generates the output of the LSTM cell denoted h_t

$$h_t = \tanh(C_t) \cdot u_t \tag{46}$$

As figure 5 shows, this value is passed on to the next layer in the network and generated further by the layers corresponding activation function. The same value h_t together with the cell state C_t is passed on for the next iteration in the LSTM-cell, albeit now defined h_{t-1} and C_{t-1}. The addition of LSTM-cells does not affect how training of the network is done as the loss function is the same [AG].

2.3.5 Mixture Density Network

As the name suggests, the Mixture Density Network (MDN) builds on the idea of parametric mixture models and the neural network. The aim is to describe the conditional distribution of a target variable, y, as a mixture of K distributions derived from the mean and variance of the input x. The main objective here is thus to find the conditional distribution as described by the mixture model

$$P(y\,|x) = \sum_{k=1}^{K} \pi_k(x)\mathcal{N}(y\,|\mu_k(x), \sigma_k^2(x)) \tag{47}$$

Equation 47 is the case for a *Gaussian mixture model*, and consists of three main parameters $\mu_k(x)$, $\sigma_k(x)$ and $\pi_k(x)$. Where $\mu_k(x)$ is the mean, $\sigma_k(x)$ is the variance, and $\pi_k(x)$ being the mixture coefficient. The mixture coefficient can be seen as prior probabilities of the target y being generated from the *k:th* component of the mixture model. In equation 47, $\mathcal{N}(y\,|\mu_k(x), \sigma_k^2(x))$ is the estimated conditional density of the target y. As Bishop [BC] describes it, the estimation of the density is done via so-called *kernel functions*. For this project, the kernel function chosen is Gaussian as this is commonly chosen for financial data [GHS]. These are of the form

$$\mathcal{N}(y\,|\mu_k(x), \sigma_k^2(x)) = \frac{1}{\sigma_k(x)\sqrt{2\pi}} \exp\left(-\frac{||(y-\mu_k(x))||^2}{2\sigma_k^2(x)}\right) \tag{48}$$

In the MDN, the parameters $\mu_k(x)$, $\sigma_k(x)$ and $\pi_k(x)$ are all conditioned on the input x. The aim of the MDN is thus to model the parameters of all the component K densities by the outputs of a neural network. In turn, the parameters must fulfill certain constraints in order to be considered feasible. As such, the MDN layer is implemented at the output layer of the neural network where each parameter is treated on its own using separate activation functions. Effectively, this means that the MDN layer consists of three layers, one for each parameter. Thus, the input to each one of these "parameter layers", derived as the output of the network prior to the MDN layer, are below denoted as g^π, g^σ and g^μ. Moreover, denoting the number of components of the target variable y as L and the amount of mixture coefficients as K, the total amount of outputs from the MDN layer is $(L+2)K$. Regarding the mixture parameters $\pi_k(x)$, as these are interpreted as probabilities, they must sum to unity. In the network, this is obtained by using a softmax activation

$$\pi_k = \frac{\exp(g_k^\pi)}{\sum_{j=1}^{K} \exp(g_j^\pi)} \tag{49}$$

where g_k^π is the k:th element of g^π with k being the corresponding component of the mixture model and M being the size of g^π. This ensures that the values π_k lie in the range [0,1] and sum to unity.

For the variances σ_k, these will have to be greater than 0 as they are represented as scale parameters. Considering the network output, in order to get the σ_k, the output is transformed using an exponential function as such

$$\sigma_k = \exp(g_k^\sigma) \tag{50}$$

which only provides positive values. The μ_k in turn are represented directly by the network outputs g_{kl}^μ, i.e. $\mu_{kl} = g_{kl}^\mu$, where l denotes the l:th element of g_k^μ which has L components.

Regarding training of the MDN, there is a need to define a suitable loss function which considers the above mentioned parameters. This can be done using the negative log likelihood. Consider the distribution in equation 47, the negative log-likelihood, i.e., the loss function, is given by

$$\mathcal{L}(w) = -\sum_{n=1}^{N} ln\left(\sum_{k=1}^{K} \pi_k(x_n, w)\mathcal{N}(y\,|\mu_k(x_n, w), \sigma_k^2(x_n, w))\right) \tag{51}$$

where the expression for $\mathcal{N}(y\,|\mu_k(x), \sigma_k^2(x))$ in 48 holds true. This loss function is minimized by the back-propagation algorithm as defined under section 2.3. The corresponding derivatives, as brought forward by Bishop [BC] are defined by:

$$\begin{aligned}
\frac{\partial \mathcal{L}_n}{\partial g_k^{\pi}} &= \pi_k - \gamma_k \\
\frac{\partial \mathcal{L}_n}{\partial g_{kl}^{\mu}} &= \gamma_k\left(\frac{\mu_{kl} - y_l}{\sigma_k^2}\right) \\
\frac{\partial \mathcal{L}_n}{\partial g_k^{\sigma}} &= -\gamma_k\left(\frac{-||y - \mu_k||^2}{\sigma_k^2} - \frac{1}{\sigma_k}\right)
\end{aligned} \tag{52}$$

where γ_k is the corresponding posterior probability defined as

$$\gamma_k(y\,|x) = \frac{\pi_k \mathcal{N}_{nk}}{\sum_{l=1}^{K} \pi_l \mathcal{N}_{nl}} \tag{53}$$

where $\mathcal{N}_{nk}$ as in equation 48 is defined as $\mathcal{N}(y_n\,|\mu_k(x_n), \sigma_k^2(x_n))$.

Recurrent neural networks can be used effectively in unison with MDNs, allowing for a distribution conditioned on not only the current input but on a complete input sequence, enabling a more reasonable model fit with regard to the past history [MS]. LSTMs specifically have been applied successfully with MDNs by Graves [AG] and was additionally the most successful model in VaR estimation by Arimond et al. [ABHKW]. A rough schematic for the architecture in this thesis is presented in section 3.3.1.

3 Method

3.1 Outline

The theory outlined above will be used in unison for the ultimate goal of generating VaR predictions. Neural networks as such are used to capture the nonlinear dynamics of the data. Ideally, the network should consider and foresee not only current trends, but also particularly idiosyncratic events by recalling patterns that preceded these in the past. Such events are of course precisely those which are of greatest interest when discussing tail risk, i.e., the risk of extremely rare loss inducing events in the "tail" of the loss distribution. For this purpose, LSTM-layers are used in the network, owing to their penchant for handling time series data. Monte Carlo simulation of future returns will be used to derive VaR predictions, and the assumed distribution is a Gaussian mixture model. To create the mixture, a mixture density output layer is placed at the end of the network to generate the mixture probabilities $\hat{\pi}_k$ and the Gaussian parameters $\{\hat{\mu}_k, \hat{\sigma}_k\}$ for $k = 2$ regimes. This method will finally be compared with the most often used methods of predicting VaR, the historical simulation and mean-variance methods. The comparison will evaluate not only how often the VaR prediction results in an underestimation of the loss, referred to as a "breach", but to what extent the MDN can be used as a VaR model. The theory has been presented above, but the thesis specific considerations are detailed below. The following schematic is presented to briefly summarize the procedure:

1. **Data collection** and **pre-processing**.
2. Measure VaR using "classic" methods, i.e. **mean-variance approach** and **historical simulation**.
3. Construct **LSTM-MDN** to optimize ϕ_k, μ_k, and σ_k for $k = \{1,\ 2\}$ regimes by maximizing the likelihood of the posterior distribution.
   (a) Construct optimal *loss function*, potentially through regularization techniques
   (b) Choose functioning *hyperparameter architecture*
   (c) Potentially include early stopping, vary number of training epochs and further hyperparameter tuning.
4. Compare **VaR breaches/nominal exceedances** between models

3.2 Methodology

3.2.1 Data collection

The data was retrieved through GoogleFinance. Specifically, the data consists of the adjusted closing price for each day on a set of publicly traded stocks within the S&P500[4] ranging from the period of 2000-01-04 to 2020-12-30. 42 stocks were chosen randomly from a set guaranteeing values for essentially each day in the desired time period, barring a small amount of missing inputs. The number of stocks chosen was arbitrary, yet large enough to ensure a sizable data set.

[4] One of the most common stock market indices; measures stock performance of 500 large companies listed on stock exchanges in the United States.

3.2.2 Pre-processing

The data was first checked for NAs, i.e., empty values for any of the stocks on any given day in the data set, with no stock missing more than 0.3 % of its observations. Rather than remove the affected days, these values were imputed with the value for the previous day, which facilitated the next step.

The data was then converted from daily adjusted closing prices to daily returns, R_t^i with i denoting the stock and t denoting the day, following the procedure and motivation outlined in section 2.1.1. As such, the first observation from each stock's time series disappeared. Thus, the data ultimately used had the following structure:

$$\begin{matrix} \text{2000-01-04} \\ \vdots \\ \text{2020-12-30} \end{matrix} \begin{pmatrix} {R_0}^1 & \dots & {R_0}^{42} \\ \vdots & \ddots & \vdots \\ R_{5284}^1 & \dots & R_{5284}^{42} \end{pmatrix} \tag{54}$$

When using neural networks, it is often imperative to scale the data in order to speed up the learning process and achieve faster convergence. This is usually done by scaling the data to have a mean close to zero [YL]. This step is most critical when dealing with tasks containing multiple features of different scales, e.g., predicting how many children a person will have (usually in the range of 0-5) given a person's annual income (tens of thousands of dollars), height (in the approximate range of 150-200 cm) and years of post-secondary education (usually between 0-6). Given the fact that returns are already adequately centered around a mean of zero, the data at hand is essentially directly compatible with a neural network [FD], hence why no further scaling was necessitated. To note is also that no explicit variance/covariance relationship is defined as it is, if applicable, trained endogenously in the network [HPW].

Due to the mechanics of LSTM-networks, the importance of specifying an optimal lookback period, denoted d, is reduced as the network offers long and short term memory. Still, the LSTM-network does require input data to be sequential. As such, for the neural network model, the inputs were set to a size of 20, meaning that any given prediction used the past 20 days of data. The function f is a linear combination of the n portfolio stocks, which for simplicity was chosen as an equally weighted long portfolio, i.e.

$$f(\mathbf{R}_t) = \frac{1}{n} \sum_{i=1}^{n} R_t^i \tag{55}$$

Therefore, for any given day t, the input had to be configured such that x_t consisted of the returns for each of the $n = 42$ stocks for the previous 20 days, i.e. $x_t = [\mathbf{R}_{t-20}, ..., \mathbf{R}_t]^T$, where the superscript T denotes transposition. The target variable y_t was configured as the portfolio return for day $t+1$, i.e. $y_t = f(x_{t+1}) = f(\mathbf{R}_{t+1})$. In words, this implies that the model is trained to predict the portfolio return on day $t+1$ given the stock returns for days $[t-d, t]$. With $d = 20$, the input and output data for any given day t thus looked like

$$x_t = \begin{pmatrix} \mathbf{R_{t-20}} \\ \vdots \\ \mathbf{R_t} \end{pmatrix} = \begin{pmatrix} R_{t-20}^1 & \dots & R_{t-20}^{42} \\ \vdots & \ddots & \vdots \\ R_t^1 & \dots & R_t^{42} \end{pmatrix}, \; y_t = f(x_{t+1}) = \frac{1}{42} \sum_{i=1}^{42} \mathbf{R}_{t+1}^i \tag{56}$$

3.2.3 Measuring VaR classically

To benchmark the model against prevailing methods, the VaR was calculated through the mean-variance and historical simulation approaches (see section 2.2.1). After some experimentation, it was determined that these methods should use a lookback period of $d = 60$ days in order to attain sufficiently good VaR estimations. For both the mean-variance and historical simulation approaches the *Numpy* package was used and implemented in Python. In this thesis, the implemented mean-variance method required the returns over the period of interest for the *portfolio*, $[f(\mathbf{R}_{-d}), f(\mathbf{R}_{-d+1}), ..., f(\mathbf{R}_0)]^{\mathrm{T}}$ (where superscript T indicates transposition) rather than the returns for the individual assets. As such, a standard deviation σ was computed instead of a covariance matrix Σ. Using a singular portfolio variance was considered necessary to maintain comparability with the neural network approach, which is only configured to predict portfolio returns.

Thus, in the benchmark models, VaR was calculated based on a prediction of the portfolio return $V_T = f(\mathbf{R}_T)$ for a time T in the future, given a certain amount of days of prior return data $[\mathbf{R}_{T-d}, \mathbf{R}_{T-d+1}, ..., \mathbf{R}_{T-1}]^{\mathrm{T}}$, for a pre-determined number of days d. In this thesis, the value $d = 60$ was chosen after some experimentation when using the benchmark models and predicting VaR values $T = 1$ day in the future. Again, each $\mathbf{R}_t$ consists of the returns for the n assets on day t, i.e. $\mathbf{R}_t = [R_t^1, ..., R_t^n]$. VaR is thus calculated for day $T = 1$ through the historical and mean variance approach using the input $[\mathbf{R}_{-59}, \mathbf{R}_{-58}, ..., \mathbf{R}_0]^{\mathrm{T}}$. For day $T = 2$, the inputs' temporal index is incremented by 1, as the return for day $T = 1$ has been "observed" and can be used for the VaR estimate on day $T = 2$.

3.2.4 Neural network development

Implementation To construct the neural network, Keras [Keras] was used, which serves as an interface for the TensorFlow library. TensorFlow was developed by Google and is both a free and open-source library for machine learning. Keras offers both ordinary neural network layers and LSTM-layers, in addition to a multitude of different machine learning methods. For the MDN-layer, an open-source Python package called *Keras Mixture Density Network Layer* was used, available online and developed by Professor Charles Martin of the Australian National University [CM].

Loss function An example of a loss function is given in equation (27), and the mean squared error is a reasonable metric one seeks to minimize in typical neural network implementations. In this thesis, however, rather than produce point forecasts, the objective is to maximize the likelihood of the observed values being generated from the estimated distribution. As such, rather than, e.g., mean squared error, the loss function for this thesis needed to (minimize) maximize the (negative) log-likelihood; the exact loss function is described in equation (51).

In line with the findings of Arimond et al., the loss function as such tended to estimate the mixture density probabilities π_k in a non-desirable way. Specfically, one π_k invariably converged to 0 while the other converged to 1, effectively reducing the parameterization to one distribution. To mitigate this issue, inspired by Arimond et al., a regularization term was added which adds a penalty to the loss function if one-regime dominant probabilities occur. Specifically, the penalty term W was

defined as:

$$W = \sum_{k=1}^{K} \left(\frac{1}{T} \sum_{t=1}^{T} \pi_k(t) \right)^2 = \sum_{k=1}^{K} \bar{\pi}_k(t)^2 \tag{57}$$

While this thesis assumes $K = 2$ regimes, the penalty term defined above is equally functional for balancing $K > 2$ regimes. And the loss function $\mathcal{L}$ defined in equation (51) is instead replaced by $\mathcal{L}_W$ defined as

$$\mathcal{L}_W = (1 - \lambda W) \cdot \mathcal{L} \tag{58}$$

where the term $0 < \lambda < 1$ serves as a scaling factor, allowing for calibration of the penalty term, with lower values relaxing the incentive of balancing of regime probabilities and higher values increasing it. Recalling that the loss function is designed to be negative (in order for minimizing rather than maximizing to be the objective, as is required in Keras), one sees that smaller values of W results in a larger overall loss. The penalty term $W = \sum_{k=1}^{K} \bar{\pi}_k(t)^2$ is minimized for $\pi_k = 0.5$ for $k = \{1, 2\}$. To see this, first note that there are only two regimes and that for regime probabilities $\sum_{k=1}^{K} \bar{\pi}_k = 1$, implying that $\bar{\pi}_1 = 1 - \bar{\pi}_2$. W thus reduces to

$$\begin{aligned} W = \bar{\pi_1}(t)^2 + \bar{\pi_2}(t)^2 = \bar{\pi_1}(t)^2 + (1 - \bar{\pi_1}(t))^2 = \\ \bar{\pi}_1(t)^2 + (1 - 2\bar{\pi}_1(t) + \bar{\pi}_1(t)^2) = 1 - 2\bar{\pi}_1 + 2\bar{\pi}_1(t)^2 \end{aligned} \tag{59}$$

To maximize, we differentiate W and set the derivative equal to zero, which yields

$$\frac{\partial W}{\partial \bar{\pi}_1(t)} = -2 + 4\bar{\pi}_1(t) \ = \ 0 \Leftrightarrow \bar{\pi}_1(t) = 1 - \bar{\pi}_2(t) = 0.5 \tag{60}$$

Note that the second derivative is positive, thus ensuring that W is minimized:

$$\frac{\partial^2 W}{\partial \bar{\pi}_1(t)^2} = 4 < 0 \tag{61}$$

The loss function is now formulated in such a way that the likelihood of the predicted distribution generating the observations is maximized, and that the resulting regime probabilities are incentivized to achieve "realistic" or at least nonzero converging probabilities.

Hyperparameter architecture Before defining the chosen model architecture, the data was chronologically split into a training, validation, and testing set consisting of 70, 15, and 15 percent of the data, respectively (i.e. using 70 percent of the data for training the network, 15 percent for testing and 15 percent for validation). That is, the training data consisted of data from 2000-01-04 to 2014-09-23 (3684 days), the validation data consisted of 2014-09-24 to 2017-11-08 (789 days), and the test set consisted of data from 2017-11-09 to 2020-12-30 (790 days). This allows for the network to adjust its weights in accordance to the training set during training, while manual adjustments to the hyperparameters are done with the obtained training loss function value as well as the model's predictability and loss value on the validation set. The test set was left untouched and was used to test the model performance after adjustment of hyperparameters stemming from validating the model on the validation set.

As the approach for choosing the optimal neural network architecture is not rigorously defined, the process in this case largely relies on trial and error. Nevertheless, the general aim is to minimize the loss function and maximize accuracy while reducing bias and overfitting. A common approach to this is to use algorithms such as *Grid Search* where hyperparameters are iteratively chosen based on a predefined metric that the network produces (e.g. its loss). However, for this thesis, the aim is not only to minimize some metric but also to produce realistic regime probabilities, represented by π_k in the mixture model. Thus, as this objective can not be generalized into a specific metric, no algorithm was used for defining the optimal network architecture. Instead, the approach relied on extensive manual trial and error compared to an algorithm's automatic approach. Nevertheless, the hyperparameters that need to be adjusted in the neural network are the number of layers, number of nodes in each layer, choice of activation function in each layer, initializers, batch size, learning rate and number of epochs.

Layers A typical ANN has one input layer, one or more hidden layer(s), and one output layer. The most common type of layer is the *dense layer*, which, as the name suggests, is fully connected to neurons in another layer. Compared to the LSTM-layer, the dense layer is quite simple and provides learning features for all combinations from the previous layer. On its own, it does not account for temporal aspects and memory as the LSTM-layer. In this thesis, there indeed needs to exist one input layer as well as one LSTM-layer as the temporal features of the time series are essentially the only explanatory variables from which the network may perform inference on. The output layer is in this context necessarily an MDN-layer, as the output is a conditional distribution rather than a point estimate. Friedman et al state that choosing the number of hidden layers relies on the analyst's background knowledge coupled with experimentation, and that stacking of multiple layers enables the network to construct deeper features from which the output can be based on [HTF]. Gu et al noted that sparser models with fewer layers outperformed their denser counterparts in the context of financial forecasting, which was assumed to be the case due to the low signal to noise ratio in financial data [GKX]. Inspired by this fact, an effort was made to not complicate matters unnecessarily with more layers, while simultaneously adhering to established research within the field.

Nodes Friedman et al. state that in general, it is better to have too many nodes in a layer than too few [HTF]. With too few neurons, one risks missing important nonlinear connections in the data whereas the issue of having too many neurons can be overcome with various regularization techniques. Boyd and Kaastra give numerous examples of potential rules of thumb that could theoretically be used for choosing the number of neurons in the hidden layer, including (for a three-layer network with n input neurons and m output neurons): $\sqrt{n \cdot m}$ (Masters, 1993), $0.75 \cdot n$ (Baily and Thompson, 1990), and between one half to three times n (Katz, 1992) [B&K]. In any case, Friedman, Boyd & Kaastra and the vast majority of research ultimately refer to experimentation within the context of the specific problem to be solved when determining the number of neurons in hidden layers. As such, different numbers of neurons were tested until relatively encouraging results were noticed, whereupon the encouraging set-up was varied locally to further optimize the network structure.

Miscellaneous improvements One important aspect in formulating the architecture of the neural network is the usage of callbacks. A callback is a function that is applied at given stages in

the training of the neural network [KCB]. In this thesis, the callbacks were used for three purposes; automatic adjustment of learning rate, early stopping and model checkpoints. Firstly, for the learning rate, the callback *ReduceLRonPlateau* monitors the loss function and automatically reduces the learning rate by a factor of 0.1 if the loss has not improved over 15 epochs. In this callback the factor of which the learning rate is to be reduced, the number of epochs before adjustment (i.e. patience) as well as the variable that is to be monitored can be manually adjusted. Secondly, the callback *EarlyStopping* was used to introduce early stopping. This callback monitors the validation loss and stops the training of the neural network if the validation loss has not improved over 60 epochs. Similar to *ReduceLRonPlateau*, the variables in this callback can be manually adjusted as well. Lastly, the callback *ModelCheckpoint* was used to save the best model during training. When using early stopping with a patience ≥ 1, there is a risk that the model will become worse after each epoch due to suboptimal updates of the weights. Thus, it is important that the model saved and used for prediction is the model that resulted in the best loss. In doing so, when the validation loss value has been improved during training, the model weights for that epoch are saved and later overwritten if the loss is improved during later stages in the training. When the training is finished, the saved model, i.e., the best model, is then loaded and used for prediction.

Moreover, the model uses dropout layers that are put in between other layers of the model. Given a set frequency in [0,1], the layer sets a fraction of the input units to 0 at each step during training corresponding to this frequency. This means that some weights are excluded from updates while training a network. As such, the dropout layers help prevent overfitting. Moreover, inputs not set to 0 are scaled by $\frac{1}{1-frequency}$ in order to maintain that the sum over the inputs remains the same [KDO]. In the model used for this thesis, each LSTM layer is followed by a dropout layer with a frequency of 0.2, the default value in Keras dropout layers. To note is that other frequency levels were tested but yielded worse results.

3.2.5 Evaluating the models

The purpose of VaR is to estimate what a portfolio could potentially lose at a certain confidence level α. Again, a one week VaR_α of, e.g., 0.07 means that the portfolio value could decline by 7 % or more in one week with probability α. The corresponding amount of capital could then be invested in a risk-free asset (e.g. a government bond) to ensure that there is enough capital to cover any outstanding debt should the portfolio result in the potential predicted loss. If the actual loss instead exceeds the VaR estimate, a so-called "breach" has occurred and the investor might then have too little capital to cover the incurred loss. We thus to some extent compare models by the amount of breaches that each model has. However, by the very construction of VaR, if the confidence is specified to e.g. $\alpha = 0.05$, the model should breach 5 % of the time. To facilitate fair comparison among models, one must therefore check that the ratio of breaches to predictions is approximately 5 %. Given that this is the case, one should additionally consider the nominal predictions important when comparing the models. Even if two models breach roughly $100 \cdot \alpha$ % of the time, one would prefer the model that generates VaR predictions closer to the true loss. If one model gives a VaR estimate of 0.20 and another gives an estimate of 0.15 while the true loss is 0.10 (thus implying that neither model results in a breach), one would prefer the second model, as one would need to invest less money in the risk-free asset and instead be free to make further risky investments, while still being able to cover the loss of 0.10.

We let $I(A)$ denote the indicator function, i.e., a function which returns 1 if the event A occurs and 0 if not (see equation 14). The true loss is given by L while the predicted loss (VaR-estimate) be given by $\hat{L}$. The total number of predictions is given by N. With this notation, we introduce the following metrics used to investigate superiority among VaR models:

$$\text{Breach Ratio} = 100 \cdot \frac{\sum_{i=1}^{N} I(\hat{L}_i < L_i)}{N} \ (\%) \tag{62}$$

Again, if the VaR is specified at confidence level α the breach ratio should be roughly equivalent to α.

$$\text{Sum if breach} = \sum_{i=1}^{N} I(\hat{L}_i < L_i)(L_i - \hat{L}_i)$$

Here one yields the value of how much the model breaches nominally over the total amount of predictions. For models with an equivalent breach ratio, the model with a lower *Sum if breach* is preferred, as that would imply that this model is at least closer to the true value of the loss.

$$\text{Sum if no breach} = \sum_{i=1}^{N} I(L < \hat{L}_i)(\hat{L}_i - L_i)$$

Comparing *Sum if no breach* among models thus gives an indication of how close the VaR estimates are to the true loss given that the models do not breach, which again is interesting to look at as needing to invest less in the risk-free asset enables the investor to use this money elsewhere, or at the very least maintain a higher liquidity.

For general purpose comparison, we additionally look at these metrics:

$$\begin{gathered}\text{Avg. if breach} = \text{Sum if breach}/N \\ \text{Avg. if no breach} = \text{Sum if no breach}/N \\ \text{Max. VaR} = \max(\hat{L}_1, ..., \hat{L}_N) \\ \text{Min. VaR} = \min(\hat{L}_1, ..., \hat{L}_N)\end{gathered}$$

3.3 Developed models

For all models, a batch size of 32 was used. Each model was allowed to run for 500 epochs, but the final number varied based on whether or not early stopping was activated. All models had their weights randomly initialized and used the Adam optimizer. After training, the models were then fit to use for VaR simulation in line with the Monte Carlo procedure defined in section 2.2.2 to be compared with the established VaR measuring approaches.

3.3.1 Model 1 (non-regularized)

Model 1 was constructed as follows:

[LSTM-layer, 2 nodes, activation ReLU] - [Dropout layer with frequency 0.2]

[Dense layer, 12 nodes, activation ReLU] - [Dropout layer with frequency 0.2]

[MDN-output layer, activation *softmax* for π_k, and activation *ELU plus one* for σ_k].

The architecture is visualized in figure 6. The model's loss function is as given by equation 51, and is as such not incentivized to approach mixture probabilities of 0.5. While model 1 showed encouraging reactivity in predicting returns, the $\hat{\pi}_k$'s were relatively static, with one regime being invariably dominant. Model 2 was developed to generate more variability in the mixture probabilities.

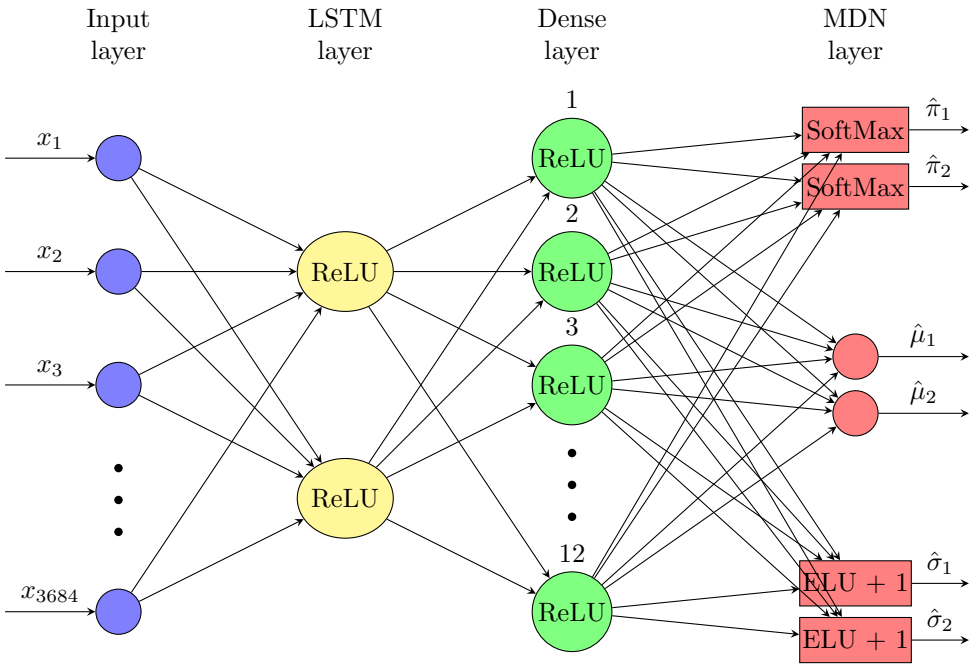

Figure 6: The architecture used in the neural network model.

3.3.2 Model 2 (regularized)

Model 2 thus used the same architecture as model 1 but with a loss function incorporating the regularization term as proposed by Arimond et al., resulting in the loss function given by equations 57 and 58.

3.3.3 Model 3 (combined)

Model 3 used the same architecture as model 1 but the mixture probabilities generated by model 2 in order to capture the relative advantages of each model.

4 Results

Model 1 is the model which had VaR breaches closest to 5 %, whereas Model 2 yields more realistic mixture probabilities (π_k). Both models suffer from a tendency to overestimate the 95 %-VaR, but exhibit qualities which in unison could work more effectively to accomplish the task. Model 3 thus uses the $\hat{\mu}_k$'s and $\hat{\sigma}_k$'s from model 1 and the mixture probabilities $\hat{\pi}_k$ from model 3. The tendency to overestimate is prevalent irrespective of confidence level chosen, i.e., estimating the 10, 5, or 1 % VaR results in the model generally overshooting its target. For this reason, focus was devoted to only presenting the 5 % VaR, as the results were deemed illustrative of the model's performance.

4.1 Model 1

In the figure below, 1-day 95 % VaR estimates are plotted for the thesis model(s) ("NNVaR X") and the chosen benchmark models, i.e. mean-variance ("ParVaR") and historical simulation ("HSVaR"). Note that only the (absolute value of) negative returns are shown as positive returns would imply that any VaR estimate will not result in a breach, i.e., only days for which the portfolio generated a loss are shown. The left y-axis shows the loss associated with the VaR estimate while the right y-axis shows the number of breaches. The dashed lines with frequent plateaus are associated with the breaches, and subsequently the higher up a line is, the more VaR breaches occurred.

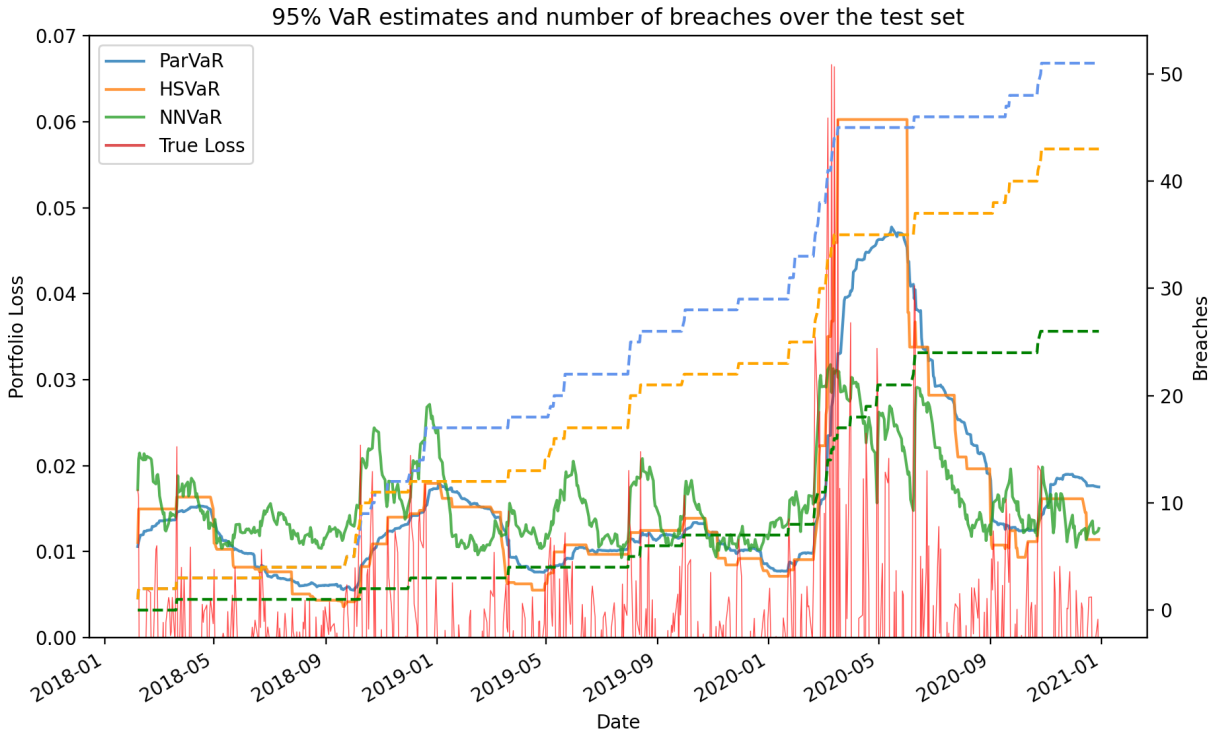

Figure 7: 1-day 95 % VaR estimates for the neural network (1), parametric, and historical simulation models computed over the test set.

What is immediately striking is the significantly fewer amount of breaches for the neural network model, being almost half of the size of the benchmark models. However, the neural network consistently "overestimates" the loss which naturally results in fewer breaches, which for practical use would entail putting away more cash than necessary to hedge for negative portfolio returns, resulting in less liquidity.

As could be inferred from figure 7, Model 1 achieves a much lower breach ratio than the benchmark models, which, as mentioned, comes at a practical cost of worse liquidity. This is evident when investigating the average VaR, as well as the *Sum if No Breach* which is significantly higher than for the benchmark models. Consequently, the *Sum if Breach* is lower, as well as the *Avg. if No Breach*.

In the next figure, the parameters of the mixture model are shown for each time t in the test set. π_1 is here thought of as the bear market and consequently π_2 is thought of as the bull market, as their corresponding normal distribution parameters show the associated characteristics of such markets, i.e. high (low) volatility and low (high) expected returns.

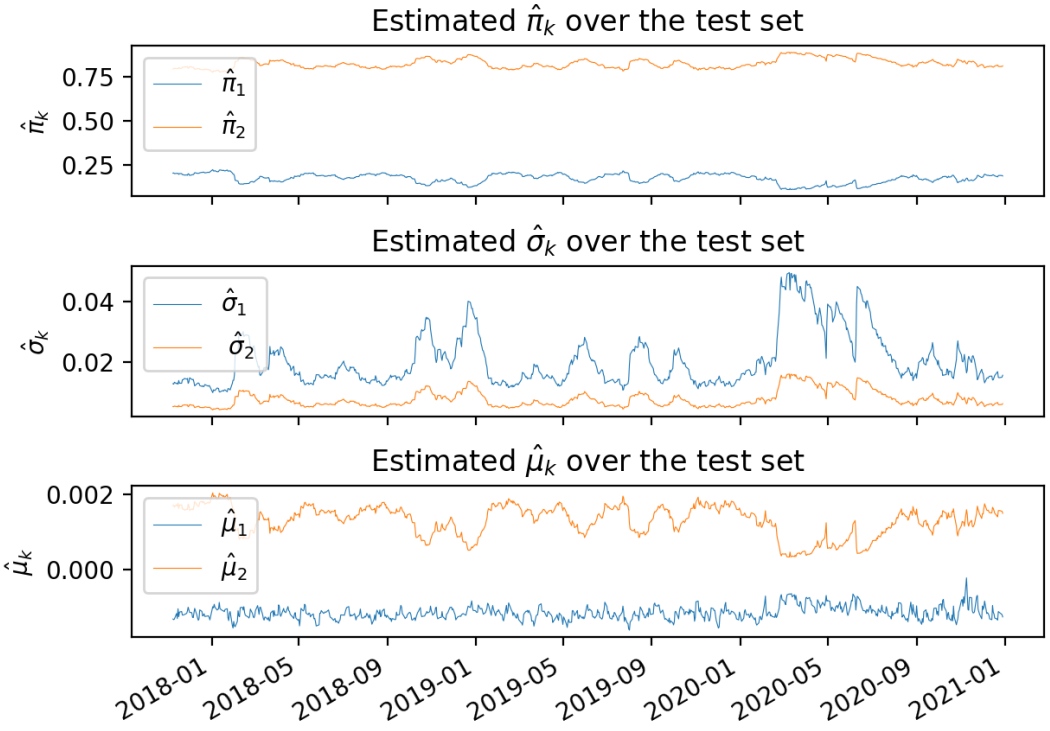

Figure 8: Estimated mixture parameters from the neural network model (1) based on the test set.

The bull market is clearly dominant as it is greater than 0.5 for each time t. The $\hat{\sigma}_k$ results reflect the volatility associated with the bear market, being considerably larger than their counterpart, the

bull market sigmas. Again one sees the regime characteristics in the sense that the bull market's $\hat{\mu}_2$ are consistently positive while the bear market's $\hat{\mu}_1$ are consistently negative. Moreover, the bull market returns are significantly more volatile, as exhibited by the higher $\hat{\sigma}_1$.

4.2 Model 2

In the second model, the loss function was configured with a regularization term, which induces the model strive for more balanced mixture probabilities.

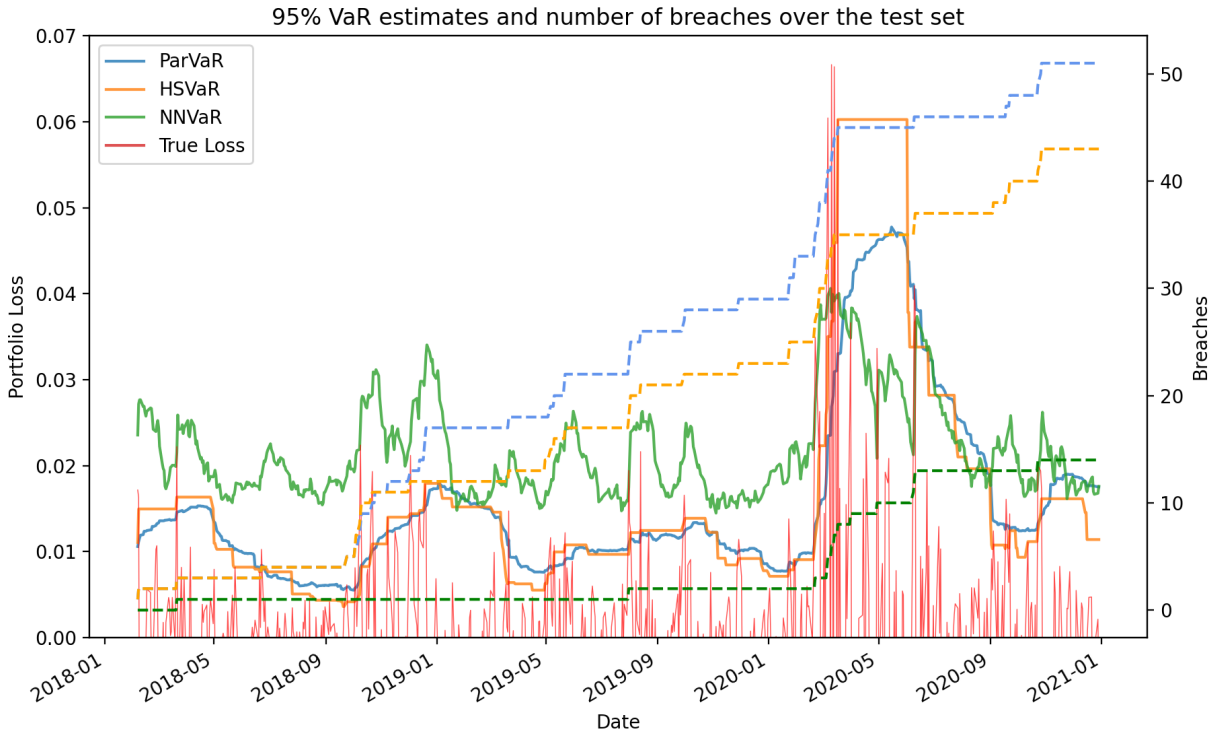

Figure 9: 1-day 95 % VaR estimates for the neural network (2), parametric, and historical simulation models computed over the test set.

Model 2 also overestimates the portfolio loss, and to a more severe degree than model 1. Of course, this results in even fewer breaches but again at the practical cost of poor liquidity.

Figure 10: Estimated mixture parameters from the neural network model (2) based on the test set.

In model 2, the mixture probabilities are piecewise dominating, thus achieving the regime switching capabilities desired. What is problematic is that one would assume that the market crash in the first quarter of 2020 would be characterized as a bear market, yet the model classifies it as a bull market. That is, for that period, the dominating $\hat{\pi}_2$ has a corresponding low $\hat{\sigma}_k$ and high $\hat{\mu}_k$ which intuitively seems wrong. However, one should note that during the market crash in the first quarter of 2020, the associated $\hat{\sigma}_2$ is increased and $\hat{\mu}_2$ is decreased.

4.3 Model 3

Model 3 uses the normal distribution parameters of model 1 and the mixture probabilities of model 2; the bear market parameters are here associated with the mixture probability dominating during the Corona crash, in order to achieve the desired characteristics of the regimes.

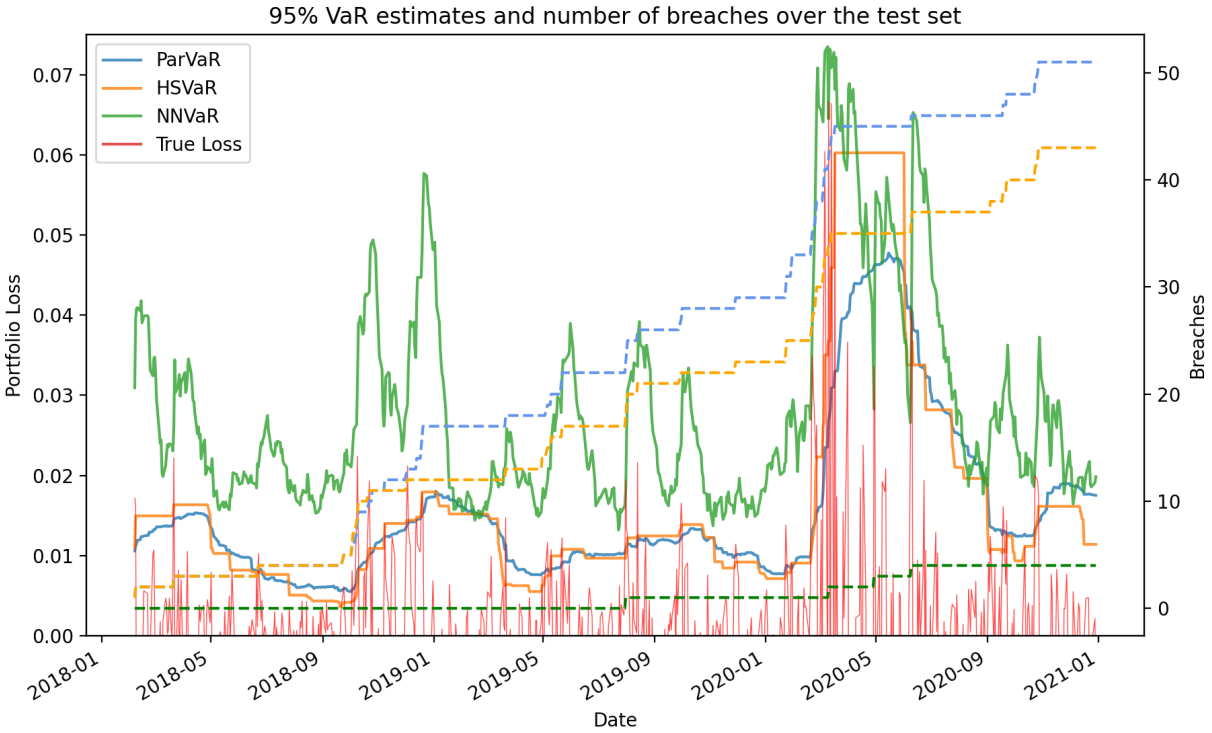

Figure 11: 1-day 95 % VaR estimates for the neural network (3), parametric, and historical simulation models computed over the test set.

Model 3 also exhibits overshooting, and to a somewhat larger degree than the previous models, however its reactivity to large losses is encouraging.

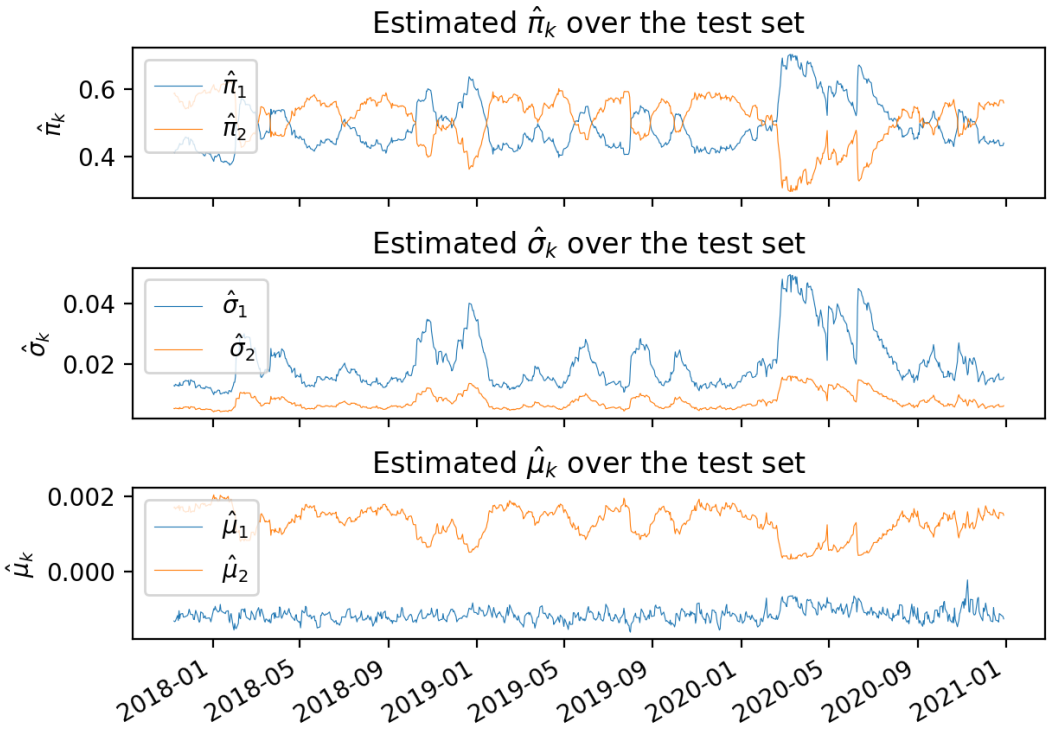

Figure 12: Estimated mixture parameters from the neural network model (3) based on the test set. These are the $\hat{\pi}_k$ from model 2 (figure 10) and $\hat{\sigma}_k$, $\hat{\mu}_k$ from model 1 (figure 8).

The manual configuration now results in the parameters exhibiting the desired regime characteristics.

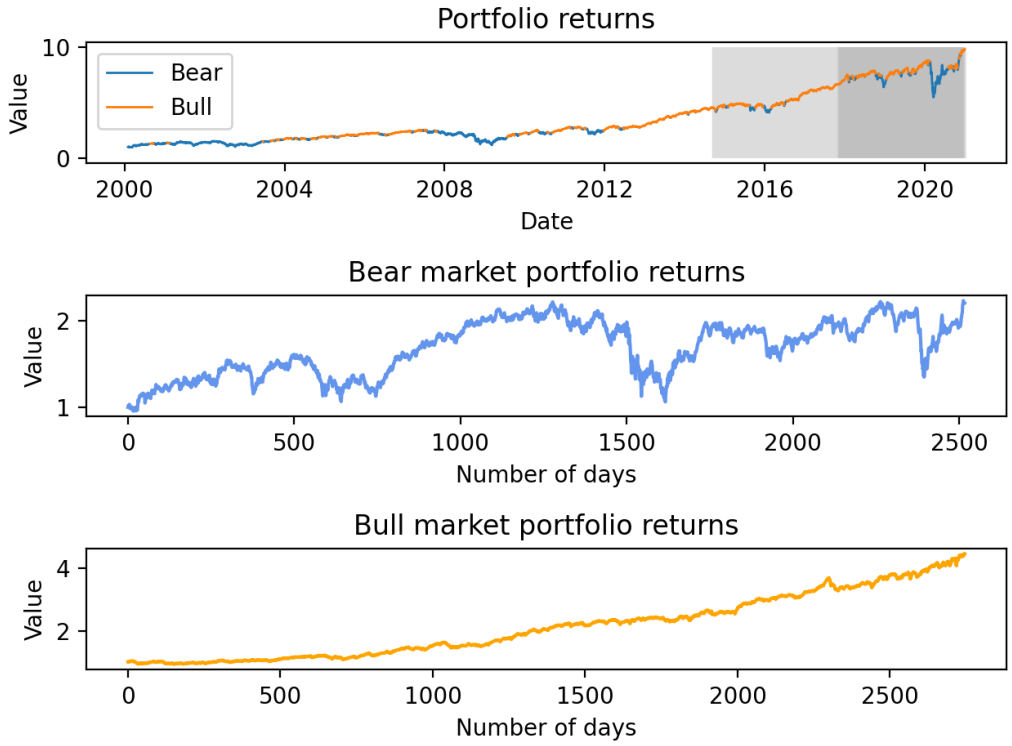

Figure 13: Portfolio value path given a starting wealth of 1 currency unit for each regime and the true returns. For the first panel, the colored areas correspond to the training, validation, and test sets respectively.

In the two lower panels in the figure (13) above, the returns from each regime are aggregated chronologically and multiplied by a starting wealth of 1 currency unit, thus showing the trajectory of a portfolio undergoing only a bear or bull market, respectively. To exemplify: suppose days 1, 3, and 5 were classified as a bear market and days 2, 4, and 6 were classified as a bull market; letting the return for day t be denoted by R_t, the fictional portfolio value 1 is multiplied by $1+R_1, 1+R_3$, and $1+R_5$ for the bear market and $1+R_2, 1+R_4$, and $1+R_6$ for the bull market. Since the days labeled as bear and bull markets are possibly disjunct, the x-axis for those panels only shows the total number of days in which either a bear or bull market reigned. The first panel shows the true returns, with each return point being colored according to the dominating regime probability. The aggregated bear and bull market returns show the desired characteristics of the regimes: the bear market is volatile, however ultimately results in a positive return on the portfolio. The bull market regime exhibits a steady increase and results in a quadrupling of the initial investment. Note that this graph aims to show the classification behaviour of the model, not its predictive ability. Hence, the results in the first panel from the training (white area) and validation set (light grey area) are shown alongside the results from the test set (dark grey area). Nevertheless, in the first panel, one sees the bear market being active where one would expect, most notably in the period of the financial crisis in 2009 and the tumultuous COVID-19 induced market crash of 2020.

4.3.1 Model 3 Shifted

Finally, we present model 3 shifted downward by the term 1.50×[Avg. if No Breach]. The Avg. if No Breach is calculated on the validation set, thus implying that the presented results are still out of sample in order to preserve generalizability of the model as such. The reasoning behind adjusting it for this difference is that the model consistently overestimates the predicted loss, and the factor 1.50 was acquired through seeking a breach ratio of roughly 5 % on the validation set.

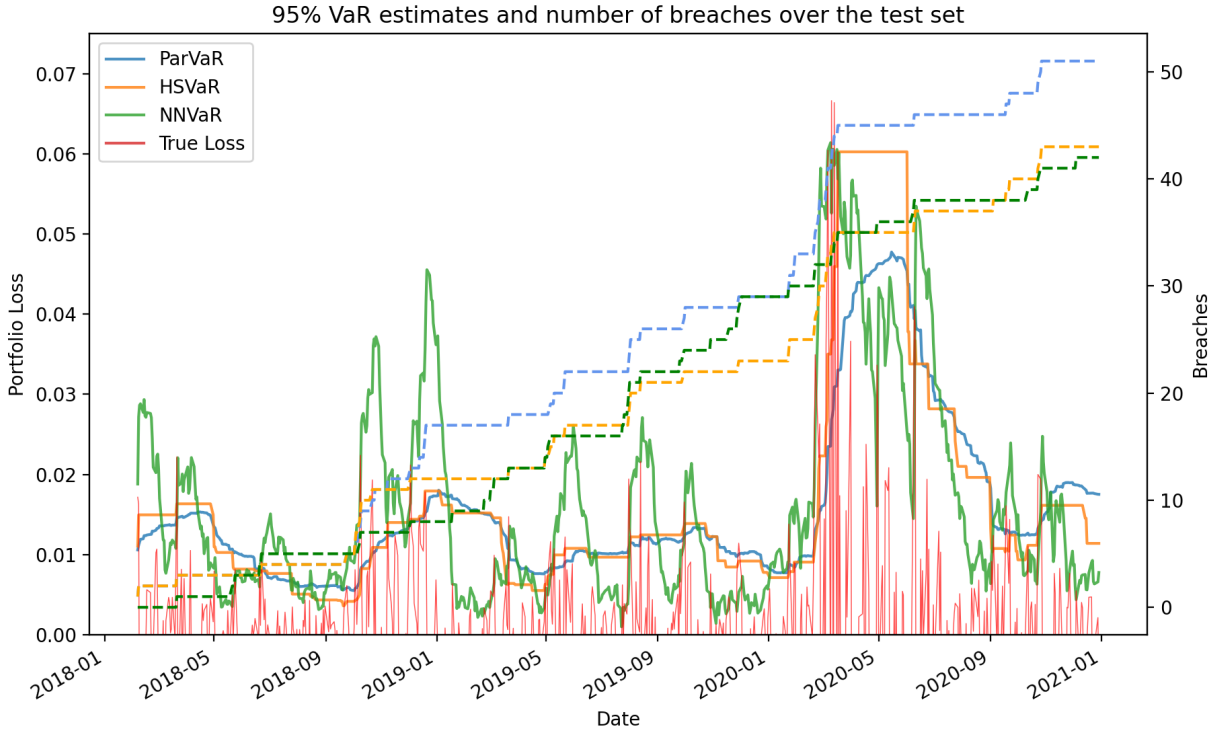

Figure 14: 1-day 95 % VaR estimates for the shifted model, parametric, and historical simulation models computed over the test set

With these adjustments, the VaR breaches are comparable to those of the historical simulation approach, while still showing significant reactivity to sudden changes in the portfolio.

Below is shown the VaR estimation over the test set for the three approaches, with the higher estimation being clearly visible for each day. The lines are the cumulative sum of the VaR estimation for each model, i.e., the sum of all VaR estimations over time on the test set. This is of interest as it shows if the usage of the NNVaR model as a risk model would result in the investor being able to take on more risk due to the precision in the VaR estimates. Specifically, between two equivalently "valid" VaR models (in the sense that both models achieve the theoretically assumed αN breaches, with sufficient independence among occurrences), the model that achieves a lower cumulative sum is superior. This is a consequence of less capital needed to be invested in the risk-free asset, enabling additional risky ventures or higher liquidity.

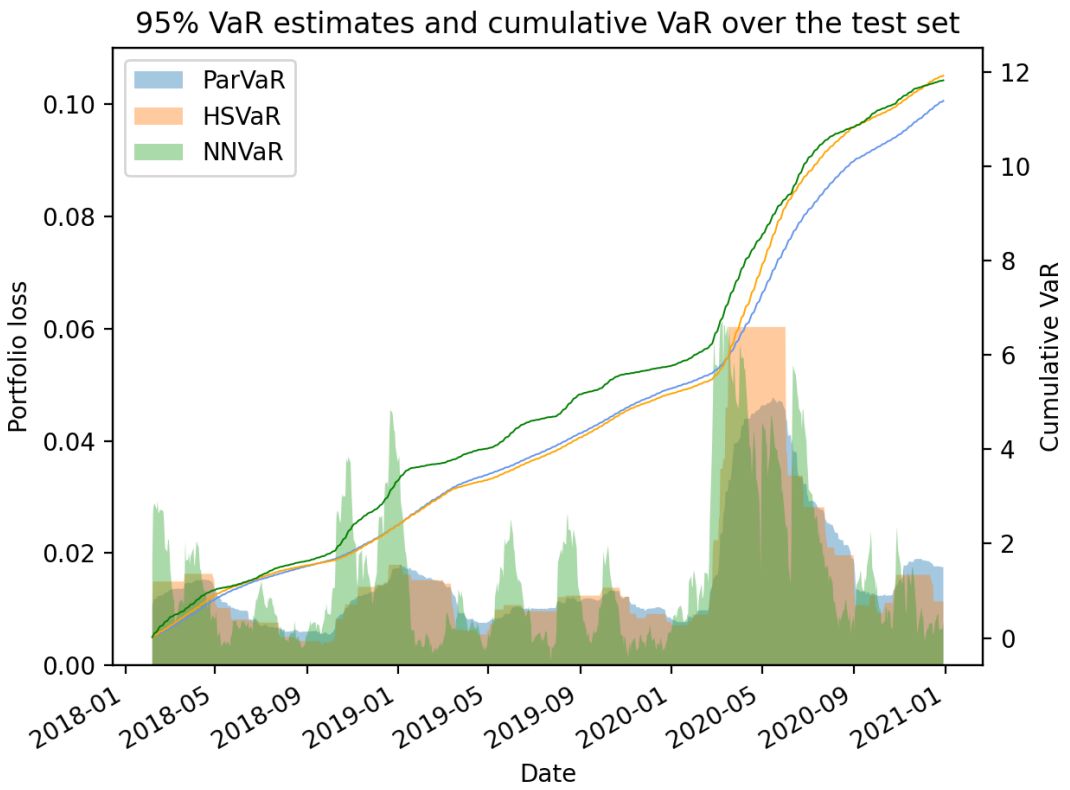

Figure 15: Cumulative 1-day 95 % VaR estimates for the shifted model, parametric, and historical simulation models computed over the test set.

In the shifted model, the cumulative sum is thus comparable to the benchmark models.

In table 1 below, important measures are presented for each model. Each measure is explained in detail in section 3.2.5.

Measure	HSVaR	ParVaR	NNVaR 1	NNVaR 2	NNVaR 3	NNVaR 3 Shifted
Breach ratio	5.89%	6.99%	3.56%	1.92%	0.55%	5.75%
Sum if breach	3.03e-01	3.88e-01	2.90e-01	1.55e-01	1.50e-02	2.00e-01
Sum if no breach	3.13e00	2.98e00	3.13e00	4.98e00	7.06e00	3.13e00
Avg. if no breach	4.29e-03	4.08e-03	4.28e-03	6.82e-03	9.67e-03	4.29e-03
Avg. if breach	4.15e-04	5.31e-04	3.98e-04	2.13e-04	2.00e-05	2.74e-04
Avg. VaR	1.63e-02	1.56e-02	1.59e-02	2.18e-02	2.85e-02	1.62e-02
Max. VaR	6.03e-02	4.78e-02	3.16e-02	4.05e-02	7.37e-02	6.14e-02
Min. VaR	3.60e-03	5.51e-03	9.28e-03	1.44e-02	1.33e-02	9.75e-04

Table 1: Statistics for the models over the test set.

We see from table 1 that the breach ratios for the nonshifted neural network models are alarmingly lower than the expected 100α %, which from the corresponding figures (7, 9, 11) is evident as the estimations lie significantly higher than the observed losses. While *Sum if breach*, *Sum if no breach*, and *Avg. VaR* are relatively comparable, figure 7 reveals that the early 2020 period may be the reason that these balance out, despite the much lower breach ratio, which can be attributed to the more stable periods in which NNVaR 1 overshoots.

In table 2 below, results from the different VaR model validation tests are shown for each model. As explained in section 2.2.3, these tests aim to measure the VaR models' validity in terms of number of breaches and independence in their occurrence. Specifically, recall that the UC-test investigates if the realized breaches are approximately proportional to the confidence level $\alpha = 0.05$; the IND-test investigates the independence of breach occurrences and the CC-test serves as an amalgam of the two. Moreover, one seeks to discard the alternative hypothesis for each of the tests, thus meaning that the null hypothesis is accepted for p-values exceeding 0.05.

Test	HSVaR	ParVaR	NNVaR 1	NNVaR 2	NNVaR 3	NNVaR 3 Shifted
UC	5.67e-01	7.05e-02	7.38e-02	5.43e-06	1.70e-12	1.76e-01
IND	2.14e-11	4.36e-13	2.00e-05	1.77e-03	8.01e-01	9.00e-06
CC	1.56e-10	7.84e-13	3.00e-05	2.44e-07	1.48e-11	2.00e-05

Table 2: P-values from the conditional coverage test over the test set

As can be seen in table 2 the final model *NNVaR 3 Shifted* performs similarly to the benchmarked models *HSVaR* and *ParVaR*. Although no model passes the CC-test, which might be due to the clustering of breaches around the Corona crisis in the beginning of 2020, what is important is that the developed NNVaR model displays similar or even better validity than the established models.

5 Discussion

The following section is divided into three subsections discussing first the results and their implications, followed by a discussion on the practicality of applying the method outlined in this thesis and finally a problematization of the usage of neural networks for financial purposes.

5.1 Model results

5.1.1 Reactivity

Looking at the model predictions above, the constructed neural network model is quite reactive in comparison to the established models (historical simulation and the parametric model). Considering the construction of the neural network, this is not substantially surprising. While all models uses historical data to produce the distributions from which the quantiles are calculated, the neural network model constructs its distributions with the aim of matching the portfolio return for the next day. Relating this to the parametric model, the resulting distribution is produced through fitting a Gaussian distribution on historic returns within a certain window. This window is later moved forward in order to include the same amount of data points when the distribution for the next day is determined. Intuitively, this results in a "lag" in reactivity for the parametric model since a short sequence of, for example, large negative returns in a recession period will not influence the fitted distribution until the period has passed. Looking at figure 7 one can see what is explained above. During the period of the Corona crisis between March and May of 2020, the parametric model (ParVaR) lags behind and predicts it largest losses during May while the largest losses of the crisis had already occurred during March. This behaviour is similar throughout the test set and as a result, the parametric model sees an increase in breaches during periods when volatility is high. A similar behaviour can be seen in the historical simulation model. The historical model is however quicker to react to the Corona crisis. Although the two models follow a similar way of construction, the historical model uses the empirical distribution over historical returns to determine its quantile. As a result of this, the tail values are directly impacted by the introduction of large historical portfolio losses and its capability to be more reactive is therefore greater than that of the parametric model.

Furthermore, due to this "lagged" behaviour of the two established VaR-models, they have the capability of being beneficial during longer periods of low volatility. Looking at the period of May to September of 2018, the model's VaR-predictions are close to the true loss of the portfolio as a result of this. Nevertheless, this type of "lagged" behaviour is not visible for the neural network model since it does not adapt its produced distributions solely on the historical returns but rather to better fit the predicted portfolio return. As a result of this, the produced distribution can be quite different from one day to another. Looking at the parameters in figure 8 one can see that the $\hat{\mu}_k$ and $\hat{\sigma}_k$ varies quite frequently throughout the test set. Relating this to the parametric model, its parameters would not fluctuate as much and the derivative throughout its parameter graphs would be more consistent. As explained above, one can see in figure 8 that during the periods of which the estimated volatility, $\hat{\sigma}_k$, is low, the parametric and historical models perform well.

5.1.2 Dominating regimes

When the model was not regularized, the MDN-model invariably let one regime become dominant for all t, an undesirable result considering the objective of creating a reactive risk model sensitive to the current market regime. While the model's prediction for any given day is based on the current input, it is possible that the resulting weights from the training have been configured in a way that gives the most appropriate mixture model for the entire data set. Over the past twenty or so years, despite occasional economic crises the market has undoubtedly been bullish, a fact visible in the last plot in figure 13. As such, with no regularization term, the model may strive for as general a mixture model as possible, which for the training data would be an overwhelmingly bullish market.

In Bishop's original MDN-paper the idea of regularization terms in the error function was put forth as a way to increase generalization of results. When applying the term as suggested in Arimond et al the problem of dominating regimes was alleviated, giving time varying dominance as seen in figure 10. When incorporating the regularization term unscaled, the mixture probabilities centered around 0.5 each, whereas scaling the term with a factor $\lambda = 0.1$ (effectively relaxing the balancing incentive) yielded the $\hat{\pi}_k$'s presented in figure 10. Still, in both cases, one would preferably like to see even larger discrepancies between the probabilities, as certain periods are undoubtedly bull or bear markets. More problematic however is the fact that for the regularized model (2), the $\hat{\mu}_k$'s and $\hat{\sigma}_k$'s do not correspond with the prior assumptions on when the bull or bear market would reign. For example, during the Corona crash one would expect the high volatility regime (bear) to be dominant, yet as seen in figure 10 the higher return/lower volatility regime's probability is higher. That fact inspired the combined model (3; see next section), but in any case when aggregating the returns as shown in figure 13 one sees that expected bear markets are at least codified the same, e.g. early 2000s IT bubble, the 2008 financial crisis and the Corona crash of early 2020. Moreover, the true returns of these periods reflect the expected characteristics, i.e. the pure bull market graph is steadily increasing while the pure bear market graph is significantly more jagged.

The results are thus somewhat encouraging, potentially motivating more refined regime classification models not necessarily through MDNs. Hidden Markov Models are commonly used for regime classification as well as parameterization of regime distributions, but for pure regime classification in a machine learning framework, a suggestion would be to include more features and try methods other than neural networks with more lenient data requirements, such as random forests or gradient boosting techniques.

5.1.3 The combined model

The reason for the logical mismatch in output parameters from model 2 was examined but no real conclusions could be drawn. Nevertheless, as the probabilities $\hat{\pi}_k$ showed the sought after regime switching characteristic, it was decided to extract the $\hat{\pi}_k$:s from model 2 and replace the non-switching $\hat{\pi}_k$:s in model 1. Thus, model 2 was used more as a classifying model rather than a loss prediction model. With this interpretation of model 2, it was deemed relevant to extract the $\hat{\pi}_k$ despite the fact that the $\hat{\pi}_k$ were generated in unison with the remainder of the mixture parameters since they showed a promising classifying behaviour of the regimes. Moreover, the $\hat{\mu}_k$ and $\hat{\sigma}_k$ in figure 10 and figure 8 are very similar, which in turn ameliorates the downfalls of combining the two models in this regard. It should also be noted that the aim of this thesis is to examine the usage of MDNs for VaR-estimation, not explicitly formulating a ready-to-use model, thus combin-

ing the outputs of the two models is of interest as it aids in examining the potential of the approach.

The results, as shown in figure 11 demonstrate an increased reactive behaviour of the model. Due to the now regime switching characteristic of the model, the part of the mixture that has higher volatility and lower estimated return has a higher probability of being sampled from during periods of higher loss. As such, the corresponding VaR-estimate is increased during these periods which explains the more jagged behaviour of the neural network model. Ignoring the model's fairly constant overshooting of its predictions, the graph portrays a behaviour that is similar to that of the true portfolio losses. As such, the idea of using a separate model to classify regimes and combining it with a model that produces the other relevant parameters of the mixture is of interest. Arimond et al use a Hidden Markov model to estimate values for the mixture parameters while letting a neural network output the $\hat{\pi}_k$. The idea here is similar, letting one network produce the $\hat{\pi}_k$ and another produce the $\hat{\mu}_k$ and $\hat{\sigma}_k$, however it should be investigated whether a neural network is best for classifying the regimes. For classification problems, there is a plethora of methods that could be used, often with more lenient data requirements and faster implementation. Since such methods would enable faster computation, one could additionally explore more features than returns, and the existence of more classes. In line with this, there are several unsupervised learning methods which may be explored, enabling the analyst to eschew preconceived notions of bear and bull markets, to identify more current trends which could potentially be more relevant for risk management.

Nevertheless, for the combined model used in this thesis, the issue of overestimating tail risk and consequently overshooting the VaR-predictions is still prevalent. While the jagged behaviour (as explained earlier) is necessary to emulate the true portfolio losses, it results in overestimating the VaR almost throughout the test set. As such, the number of breaches as shown in figure 11 is very low. However, due to this overestimation during "normal" times (e.g. the period between approx. January 2018 and January 2020) the model is able to capture the large portfolio losses during the Corona crisis where one could argue that an overestimation of losses was necessary. One can also see in the figure that, as a result of this, the neural network model suffers fewer breaches during the crisis as compared to the parametric and historical models. On the other hand, one can also argue that this is not beneficial for the purpose of the model. That is, a VaR-model at confidence level 5% should breach at approximately 5 % of the time.

5.1.4 Overestimation

The tendency to consistently overestimate the portfolio loss was a common theme among the models, both for low and high portfolio losses. There are many possible reasons for why this could be occurring. First, it should be said that the problem was persistent and seemingly unaffected by changes in the model architecture, including different activation functions, dropout frequencies, and number of layers, as well as more general data scientific configurations, such as different sizes of the training data, different seedings for the random number generator, and even choice of data set. This implies that the overestimation problem stems from the model choice rather than any particular configuration of the model. A potential reason could again be the objective of the model to reasonably fit a mixture distribution to the entire data set, and as such aims to "play it safe" by constructing distributions which include the outlier returns. That is, that the model might be biased towards the outlier returns resulting in its weights being determined so as to reduce the

outliers' impact on the loss. In other words, the model seeks an average between the error introduced by the outliers and the error introduced by the more "normal" returns. As mentioned above, multiple attempts were made to reduce the overshooting, including removing the bias vector from the layers and scaling the inputs via a min-max scaler, yet neither proved beneficial for the model. It might seem that the problem could be resolved by scaling as it would result in the model not being biased towards the large absolute portfolio returns, however the outliers only occur during a small percentage of the entire time series. As such, scaling the variable was proven to not introduce any significant results.

A second theory to why the model overestimates might be due to the usage of the $ReLU$ activation function in the LSTM-layer. As described in section 2.3.4, the *tanh* activation is typically used in LSTM-layers to scale and control the values when determining the cell state and candidate values; with the data and loss function used in this thesis, however, *tanh* produced strange and unfit VaR estimations, hence why after some experimentation the $ReLU$ function was chosen instead. As such, using a $ReLU$ activation function for this, the cell state values are not scaled in the same way and is only "controlled" by the internal *sigmoid* activations. This essentially allows for the cell state C_t to become any number larger than 0. It is thus possible that when the LSTM-layer receives input in the form of an outlier, its memory is greatly affected by this and, due to the recurrent nature of the LSTM, affects the predictions being made later on. Similarly to what was explained above, the model may become biased toward the outlier returns. Nevertheless, while this thesis found no definitive explanation architecturally for this phenomenon, bespoke alterations to the activation functions could be of interest in future research. Alterations that preferably should combat the issue of overshooting while still maintaining the beneficial regime probabilities that the $ReLU$ activation function helped produce.

Another issue could be that clear bear- and bull market distinctions are noticed on different time intervals from those which are used for VaR predictions. What is meant by this is that what is referred to as bull and bear regimes in e.g. a media context is usually not realized or noticeable on a weekly or even monthly basis, meaning that mixture distributions over short time periods may not be appropriate. Even though the bear and bull regimes in the context of this thesis are meant to accomodate VaR prediction over short periods, the multimodality needed for mixture densities to be appropriate are simply not reflected in historical returns when only looking at short time periods. As seen in figure 16 below, the normal assumption is not truly contested until at least one year returns are investigated and for a two regime case seems most appropriate the five-year return case. Even for the case of alternatingly dominant $\hat{\pi}_k$, the fact that the expected returns for either regime are extremely close implies that the mixture distribution may collapse into a unimodal set of returns, albeit with different volatilities. When deriving the quantile from the simulation, the bearish volatility will thus most likely be the determining factor, given relative balance between the mixture probabilities.

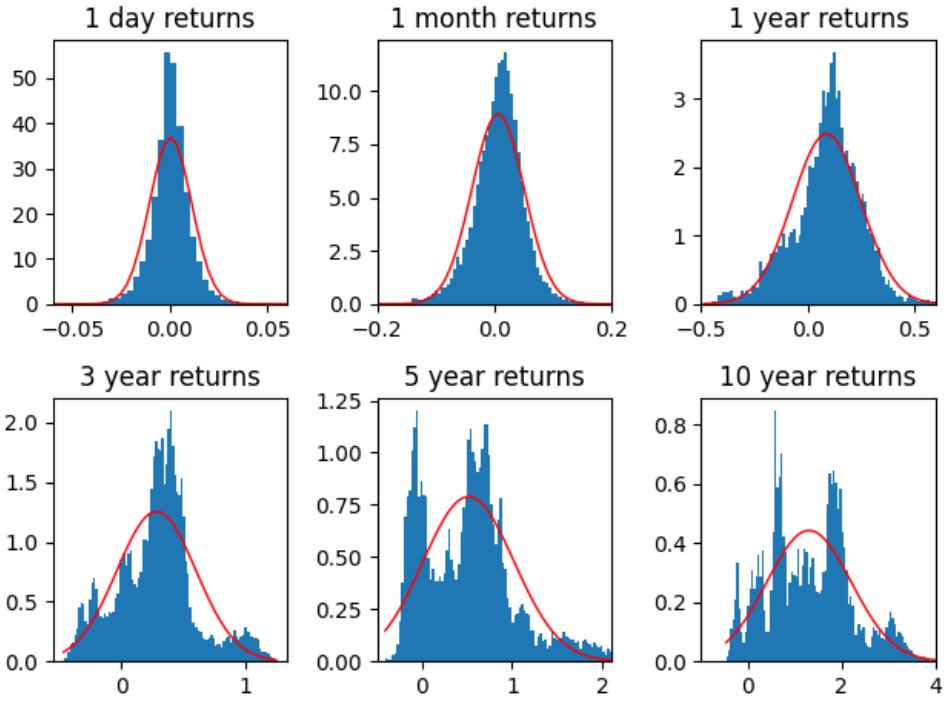

Figure 16: Histogram of rolling returns for one day, one month, one year, three years, five years and ten years of the S&P 500 index over the period 1970-2020. In blue are the true returns and the red line is the fitted normal distribution.

5.1.5 The shifted model

As explained in section 4.3.1, the combined model was shifted downward using a variable derived from the validation set in order to deal with the overestimation. The results in figure 14 show a model which follows the path of the true portfolio losses quite well and breaches approximately the same amount as the historical simulation model. Looking at the statistics presented in table 1, the breach ratio for the shifted model is close to 5 %, which is close to what Kupiec suggests should hold true for a VaR model. Looking at the test results presented in table 2, the shifted neural network model performs similarly to the parametric and historical models. The breaches are considered unconditional yet not independent over the test set. This is explained by the clustering of breaches over the Corona crisis period resulting in the rejection of the IND null hypothesis. However, as both the parametric and historical models perform similarly, the shifted neural network model is benchmarked as a satisfactory VaR model based on the test performed in this thesis. Interestingly, the combined model (NNVaR 3) manages to accept the IND null hypothesis. However, due to the combined model's amount of breaches, or lack thereof, this result is considered to be of no influence as it is expected that some clustering of breaches should exist over the test set.

Furthermore, regarding the shifted model, the *Min VaR* reaches a significantly lower minimum than the historical and parametric approaches, demonstrating not only the capacity to account for

particularly large losses but smaller ones as well. This is to some extent reflected in *Sum if breach* which is also significantly lower than the benchmark models, meaning that even if the estimation breaches, the difference will be smaller in an economically significant way. If the model does not breach, however, the aggregated sum *Sum if no breach* is more or less the same among the benchmarks and the shifted model, a fact visible in the *Avg. VaR* as well. In aggregate, these results show that the models perform comparably in terms of general performance over a longer time period, to some extent reflected in figure 15 in the sense that the cumulative sums approach the same level. The same figure also clearly highlights the more jagged behavior of the neural network model and consequently its higher reactivity. It is due to this higher reactivity that the neural network model experiences an increase in cumulative VaR during September 2018 through February 2019. A lower cumulative VaR in conjunction with few breaches would mean that the investor is able to take on more risk or alternatively maintain a higher liquidity. Although the model quickly corrects itself after this period, producing low yet accurate VaR estimations, the cumulative VaR is at this point already affected. This inspires the notion of using the established models, which are indeed faster and require less data than a neural network during calmer periods and the more reactive model developed in this thesis during more turbulent periods where the reactive behaviour is beneficial.

Future research may benefit from evaluating ways to circumvent the need to manually shift the model. Considering the success of a regularization term for the mixture probabilities, one may consider ways in which to regularize the $\hat{\mu}_k$'s and $\hat{\sigma}_k$'s so that these are more distinguished from each other, thus avoiding the collapse into a unimodal distribution. Similarly, one may investigate ways to incorporate the VaR prediction into the loss function such that the model focuses on creating reasonable quantiles rather than all-encompassing distributions.

5.2 Practicality

An important consideration, regardless of any potential success using MDNs for VaR estimation, is the practical usefulness of such an approach. In this thesis, manual configuration was required to achieve predictions which did not substantially overestimate the true portfolio loss at any given period, whereas the benchmark models give sufficient estimations with little to no configuration whatsoever. Disregarding manual configurations such as shifting the predictions, machine learning methods in general and neural networks in particular require large amounts of data to learn the patterns necessary for adequately predicting the target. In addition to low data requirements, the benchmark VaR models are near instantaneous in generating predictions, whereas the MDN approach employed in this thesis is time consuming both in the training and simulation phase. The training can of course be performed once and updated for each day at minimal computational cost, yet if one wished to employ the model for new portfolios, this would presuppose identical mechanisms to the portfolio on which the model was initially trained. Still, prevailing VaR methods tend to underestimate tail risk due to either assuming symmetry in returns (mean-variance) or relying on only historical data containing too few observations of the catastrophic events one would most like to capture in a risk management context. These considerations give credence to the possibility of applying the benchmark models in periods of relative calm, when the market sentiment is bullish or even weakly bearish, and employ MDNs or other sophisticated machine learning based approaches in anticipation of dire market downturns. Of course, such "black swan" events[5] are by definition impossible to foresee, motivating the use of machine learning and simpler methods concurrently.

[5] A metaphor for unpredictable and catastrophic events popularized by Nassim Nicholas Taleb.

Lastly, it should be said that neural networks are "black boxes", in the sense that while a function f can be approximated with arbitrary precision, no insight is given into the true nature of f. This is problematic in the context of conforming to legislature such as e.g. MiFID-II, which emphasizes transparency and protection of investors.

5.3 Neural Networks and Finance

Financial data is (in)famously hard to forecast given the myriad of influence any given event can have on market sentiment. As evinced by the market crash and subsequent upturn experienced in the first quarter of 2020, there are events which can stimulate market-wide behavior not motivated specifically by rational or data-driven considerations. Patterns learned by neural networks over a certain period of time may thus be rendered useless once new and obscure market actions occur, problematizing the notion above of using machine learning methods concurrently for the purpose of hedging against severe market events. Moreover, the efficient market hypothesis suggests that public market data is insufficient for "beating the market", which may be seen as analogous to minimizing risk, as either task invariably relies on predicting the future. Moreover, if artificial neural networks are meant to simulate the biological neural networks in the physical brain, one may question the mission to use the former for something the latter cannot actually do, i.e., predict the future development of stock prices given a large data set of historical prices. Conversely, if artificial neural networks are considered universal function approximators, one can contest the idea that there exists a robust mapping from input in the form of historical stock prices to future prices as output. Even if that would be the case, equivalent results may be equally attainable with less complex machine learning methods, e.g., gradient boosting techniques. Still, MDNs are interesting in a financial context since they do not strive for perfect point predictions but rather conditional distributions which by construction offer a confidence interval for the predicted target. This is particularly useful for risk management, where decisions are often based upon probable scenarios and the severity of those scenarios should they occur. The somewhat underwhelming results in this thesis may potentially be improved upon by including other features, such as price trends, liquidity variables, and valuation ratios. The paradigm of bull and bear market may also be challenged, and despite discouraging experimentation in this thesis and in Arimond et al, modeling $k > 2$ regimes could potentially be motivated by the fact that certain market sentiments are not sufficiently characterized by being bullish or bearish.

6 Conclusion

We conclude that recurrent mixture density networks show limited promise for the task of predicting effective VaR estimates if used *as is*, i.e. without being regularized and letting all components of the mixture distribution be generated within the network. The reason for this is that the model consistently overestimates the quantile from which the VaR prediction is made. Reasons could potentially be the behavior of one day ahead returns being relatively normal (and thus unimodal), the use of ReLU in the LSTM-layer, and the fact that the weights generated by the network strive to minimize the loss by making "safe" predictions. By regularizing the loss function of the network with regard to the mixture probabilities, one can achieve more balance between them at the cost of reduced interpretability vis-à-vis the parameters in the regimes' respective distributions. For practical use, encouraging results were achieved when manually shifting the predictions by an average of the overestimation based on the validation set. In any case, the assumption that neural networks are a viable method for VaR prediction remains dubious considering the complexity in formulating and training the model when comparing with benchmark methods over periods of relative calm. An additional detraction lies in the vast amount of data required. Moreover, considering the emphasis of transparency in regulatory frameworks intended to protect investors, the viability of neural networks becomes further reduced due to a lack of interpretability. For crisis events, however, the reactivity of the recurrent MDN approach becomes an important selling point missed by other models and motivates further studies.

Specific suggestions for future research include:

- **Explore other ways to classify regimes, through e.g. other machine learning methods with more lenient data requirements**. Focusing only on the classification aspect of this task would allow for less data intensive machine learning methods, allowing for more features to be used. Unsupervised methods could also be explored and allow for a more flexible regime paradigm.
- **Investigate why MDNs overestimate quantiles in their predictions**. While no definitive reason was identified in this thesis, potential theories were outlined in the discussion and may be validated in future work. In line with this and as mentioned, one way could be regularization of $\hat{\mu}_k$ and $\hat{\sigma}_k$. As discussed in section 5.3, financial data in and of itself may be difficult for MDNs to model, and the issue might not be prevalent for other tasks. Thus, it would be of additional interest to apply MDNs to financial data but not necessarily for use in VaR prediction.
- **Test more regimes and/or other distributions.** In this thesis only the Gaussian distribution and two regimes were tested. Although it might come at a cost of reduced economic interpretability (defining market regimes other than bear, bull or maybe "black swans") it is of interest to investigate if it would improve the model performance. Regarding distributions, using for example a *Student-t* distribution, which allows better modelling of heavy tails, might better capture outlier returns which in turn could improve the model performance.

References

[ABHKW] Arimond, Alexander; Borth, Damian; Hoepner, Andreas G. F.; Klawunn, Michael; Weisheit, Stefan. *Neural Networks and Value at Risk*. Michael J. Brennan Irish Finance Working Paper Series Research Paper No. 20-7, 2020.

[AG] Graves, Alex. *Generating Sequences With Recurrent Neural Networks*. arXiv preprint arXiv:1308.0850, 2014.

[BC] Bishop, Christopher *Mixture density networks*. Technical report, 1994.

[B&K] Boyd, Milton; Iebeling, Kaastra. *Designing a Neural Network for Forecasting Financial and Economic Time Series.* Neurocomputing 10 (1996) p. 215-236. 1995.

[CM] *keras-mdn-layer*. https://github.com/cpmpercussion/keras-mdn-layer. 2020.

[CO] *Understanding LSTM Networks*. https://colah.github.io/posts/2015-08-Understanding-LSTMs/. 2015.

[CP] Christoffersen, Peter. *Evaluating Interval Forecasts*. International Economic Review, 39, 1998, pp. 841-862.

[CPZ] Chen, Luyang; Pelger, Markus; Zhu, Jason. *Deep Learning in Asset Pricing*. Available at SSRN: https://ssrn.com/abstract=3350138 or http://dx.doi.org/10.2139/ssrn.3350138. 2020.

[CSD] Campbell, S. D. *A Review of Backtesting and Backtesting Procedures*. Finance and Economics Discussion Series, Working Paper, 2005-21.

[CWFS] Culp, Stephen; Werthen, Dr Markus; Frei, Marcus; Schelling, Robert. *Markets in Financial Instruments Directive II (MiFID II): Turning Regulatory Challenges into Business Opportunities*. Accenture, 2014.

[EF] Fama, Eugene. *Efficient Capital Markets: A Review of Theory and Empirical Work*. The Journal of Finance, 25(2), 383-417. 1970.

[FD] Daniel, Fabrice. *Financial Time Series Data Processing for Machine Learning*. Artificial Intelligence Department of Lusis, Paris, France. 2019.

[GHS] Glasserman, Paul; Heidelberger, Philip; Shahabuddin, Perwez. *Efficient Monte Carlo methods for value-at-risk*. Mastering Risk, Vol. 2, pp. 5-18. 2000.

[GKX] Gu, Shihao; Kelly, Brian; Xiu, Dacheng. *Empirical Asset Pricing via Machine Learning*. The Review of Financial Studies 33 (2020), pp. 2223–2273, 2020.

[HH] Hult, Henrik; Rehn, Carl Johan; Lindskog, Filip; Hammarlid, Ola. *Risk and Portfolio Analysis*. Springer Series in Operations Research and Financial Engineering. 2012.

[HM] Markowitz, Harry. *Portfolio Selection*. The Journal of Finance, Vol. 7, No. 1. (Mar., 1952), pp. 77-91. 1952.

[HPW] Heaton, J.B; Polson, N.G; Witte, J.H. *Deep Portfolio Theory*. ArXiv (arXiv:1605.07230v2 [q-fin.PM])

[H&S] Hochreiter, Sepp; Schmidhuber, Jürgen. *Long Short-Term Memory.* Neural Computation 9(8):1735-1780. 1997.

[HSW] Hornik, Kurt; Stinchcombe, Maxwell; White, Halbert. *Multilayer Feedforward Networks are Universal Approximators.* Neural Networks, Vol. 2, pp. 359-366, 1989. 1989.

[HT] Hurlin, Christophe. Tokpavi, Sessi. *Backtesting VaR Accuracy: A New Simple Test.* 2006. ffhalshs-00068384

[HTF] Friedman, Jerome H; Hastie, Trevor; Tibshirani, Robert. *The Elements of Statistical Learning.* Springer, 2nd ed. 2001.

[KCB] Keras. *Callbacks API.* https://keras.io/api/callbacks/.

[KD] Dowd, Kevin. *Measuring Market Risk.* John Wiley and Sons. 2nd edition. 2007.

[KDO] Keras. *Dropout layer.* https://keras.io/api/layers/regularization_layers/dropout/.

[Keras] https://keras.io/

[K&L] Kingma, Diedrik P.; Lei Ba, Jimmy. *Adam: A Method for Stochastic Optimization.* ICLR 2015. 2015.

[KP] Kupiec, Paul. *Techniques for Verifying the Accuracy of Risk Management Models.* Journal of Derivatives, 3, 1995, pp. 73-84.

[K&T] Kahneman, Daniel; Tversky, Amos. *Choices, Values, and Frames.* American Psychologist, Vol. 39, No. 4, pp. 341-350. 1984.

[KTH] Department of Mathematical Statistics. *Financial mathematics .* KTH Website. 2019.

[LRZ] Le, Quoc V; Ramachandran, Prajit; Zoph, Barret. *Searching for Activation Functions.* ArXiv (arXiv:1710.05941 [cs.NE]). 2017.

[MNPP] Mehta, Amit; Neukirchen, Max; Pfetsch, Sonja; Poppensieker, Thomas. *Managing Market Risk: Today and Tomorrow.* McKinsey Working Papers on Risk, Number 32. 2012.

[MS] Schuster, Mike. *Better generative models for sequential data problems: Bidirectional recurrent mixture density networks.* pp. 589-595. The MIT Press. 1999.

[NSS] Nixon, Kevin; Spoth, Christopher; Strachan, David. *Ten years on from the crisis.* Deloitte Financial Markets Regulatory Outlook. 2019.

[NSSZ] Nikolaev, Nikolay Y; Smirnov, Evgueni; Stamate, Daniel; Zimmer, Robert. *A regime-switching recurrent neuralnet work model applied to wind time series.* Applied Soft Computing Journal 80, pp. 723-724. 2019.

[NW] Wang, Nancy. *Spectral Portfolio Optimisation with LSTM Stock Price Prediction.* Degree Projects in Mathematical Statistics, KTH, 2020.

[SCZZ] Sun, Shiliang; Cao, Zehui; Zhu, Han; Zhao, Jing. *A Survey of Optimization Methods from a Machine Learning Perspective.* ArXiv (arXiv:1906.06821 [cs.LG]). 2019.

[YL] LeCun, Yann; Bottou, Leon; Orr, Genevieve B.; Müller, Klaus-Robert. *Efficient Backprop.* Neural Networks: tricks of the trade, Springer. 1998.