

Herman Mørkved Blom

# Estimating Value at Risk from implied volatilities using machine learning methods and quantile regression

Master's thesis in Financial Economics

Supervisor: Petter Eilif de Lange

Co-supervisor: Morten Risstad

December 2022



Norwegian University of  
Science and Technology



Herman Mørkved Blom

# **Estimating Value at Risk from implied volatilities using machine learning methods and quantile regression**

Master's thesis in Financial Economics  
Supervisor: Petter Eilif de Lange  
Co-supervisor: Morten Risstad  
December 2022

Norwegian University of Science and Technology  
Department of Economics





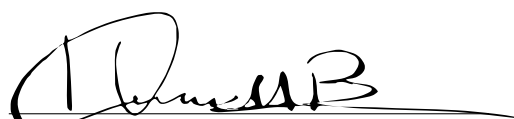
---

## Preface

This thesis marks the completion of the 2-year master program in Financial Economics at the Norwegian University of Science and Technology, NTNU, in Trondheim.

I would like to use this opportunity to thank the Department of Economics for a firm and stable learning environment, both at bachelor's and master's level. I would also like to thank my two supervisors, Petter Eilif de Lange and Morten Risstad, for excellent and rewarding feedback during the work.

Oslo, December 1, 2022

A handwritten signature in black ink, appearing to read 'Herman Mørkved Blom', with a long horizontal flourish extending to the right.

Herman Mørkved Blom



---

## Abstract

In this study we propose a semi-parametric, parsimonious Value at Risk forecasting model based on quantile regression and machine learning methods, combined with readily available market prices of option contracts from the over-the-counter foreign exchange rate interbank market.

Value at risk (VaR) is an important risk measure now widely used by financial institutions. Tools for measuring VaR with high accuracy is therefore important to assess risk in financial markets. This study attempts to improve existing methods for predicting VaR using machine learning. We use two different methods, ensemble methods and neural networks.

Explanatory variables are implied volatilities with plausible economic interpretation. The forward-looking nature of the model, achieved by the application of implied volatilities as risk factors, ensures that new information is rapidly reflected in Value at Risk estimates.

The proposed models achieves good estimates across all quantiles. Among the ensemble models, the light gradient-boosting machine model and the categorical boosting model stand out compared to our base line QR-IM model. Both models yield estimates which are better or equal to those of the bench mark model. The neural network models are in general quite unstable and could benefit from more training data and perhaps a better model architecture. Model stacking and hyperparameter tuning contribute to an improved model in terms of the over all performance of the predictions.

---





---

## Sammendrag

I denne studien foreslår vi en semi-parametrisk, sparsommelig Value at Risk-prognosemodell basert på kvantilregresjon og maskinlæringsmetoder, kombinert med markedspriser på opsjonskontrakter hentet fra over-the-counter valutakurs interbankmarkedet.

Value at risk (VaR) er et viktig risikomål som nå brukes mye av finansinstitusjoner. Verktøy for å måle VaR nøyaktig er derfor nødvendig for å kvantifisere risiko i finansmarkedene. Denne studien forsøker å forbedre eksisterende konsepter og ideer for å forutsi VaR ved hjelp av maskinlæring. To ulike konsepter brukes, ensemblemetoder og nevralt nettverk.

Implisitte volatiliteter brukes som forklaringsvariabler. Modellens fremtidsrettede natur, oppnådd ved bruk av implisitte volatiliteter, sikrer at ny informasjon raskt fremkommer i Value at Risk-estimatene.

De foreslåtte modellene oppnår gode estimater på tvers av alle kvantiler. Blant ensemblemodellene skiller light gradient-boosting machine modellen og categorical boosting modellen seg ut sammenlignet med QR-IM-modell. Begge modellene gir estimater som er bedre eller lik referansemodellen. De nevralt nettverksmodellene er generelt ganske ustabile og kan ha nytte av mer treningsdata og kanskje en bedre modellarkitektur. Modellkombinasjoner og optimalisering av hyperparameter bidrar begge til en kollektiv forbedring av predikasjonsnivåen til modellene.

---



---

# Table of Contents

<b>List of Figures</b>	<b>ix</b>
<b>List of Tables</b>	<b>x</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Literature review</b>	<b>2</b>
<b>3 Data</b>	<b>4</b>
3.1 Spot-rate returns . . . . .	4
3.2 Investigating the data . . . . .	4
3.3 Testing the data set . . . . .	4
<b>4 Methods</b>	<b>6</b>
4.1 Generating the training data . . . . .	6
4.1.1 The quantile regression implied moments model . . . . .	6
4.2 Value-at-Risk . . . . .	7
4.3 Ensemble learning . . . . .	8
4.4 Deep learning . . . . .	8
4.4.1 Artificial neural network . . . . .	9
4.4.2 Activation functions . . . . .	9
4.4.3 Training a neural network . . . . .	10
4.4.4 Optimizers . . . . .	10
4.4.5 Batch size and epochs . . . . .	11
4.5 Pre-processing . . . . .	12
4.6 Implementation . . . . .	12
4.7 The chosen ensemble models . . . . .	12
4.8 The considered neural networks . . . . .	13
4.9 Hyperparameter architecture . . . . .	14
4.9.1 Neural network hyper parameters . . . . .	14
4.10 Explainable AI . . . . .	15
4.11 Developed neural network models . . . . .	15
4.11.1 Feed forward neural network . . . . .	15
4.11.2 Recurrent neural network . . . . .	16
4.11.3 Long short term memory neural network . . . . .	16
4.12 Evaluating the models . . . . .	17

---

---

<b>5</b>	<b>Results</b>	<b>18</b>
5.1	Estimating the quantile regression VaR model . . . . .	18
5.2	Ensemble methods . . . . .	19
5.2.1	Random Forest . . . . .	19
5.2.2	Extreme Gradient Boosting . . . . .	21
5.2.3	Light Gradient-Boosting Machine . . . . .	22
5.2.4	Categorical Boosting . . . . .	26
5.2.5	Stacking . . . . .	27
5.3	Neural network . . . . .	28
5.3.1	Feed forward neural network . . . . .	28
5.3.2	Recurrent neural network . . . . .	29
5.3.3	Long short-term memory . . . . .	30
5.4	Summary . . . . .	32
<b>6</b>	<b>Conclusion</b>	<b>34</b>
	<b>References</b>	<b>35</b>

---

---

## List of Figures

1	EURUSD spot exchange rates (left panel) and daily log-returns (right panel). Time period;2009:01-2020:09. Source; Bloomberg. . . . .	5
2	EURUSD spot exchange rates (left panel) and daily log-returns (right panel). Time period;2009:01-2020:09. Source; Bloomberg. . . . .	5
3	Activation functions . . . . .	10
4	A long short term memory layer. . . . .	13
5	The feed forward neural network used in this thesis. . . . .	16
6	The recurrent neural network used in this thesis. . . . .	16
7	The LSTM neural network used in this thesis. . . . .	17
8	Quantile regression coefficients QR-IM . . . . .	18
9	Ensemble training period and validation period . . . . .	19
10	Random forest model. The black series is the daily log returns . . . . .	20
11	Random forest variable importance . . . . .	20
12	XGBoost model 1. The black series is the daily log returns . . . . .	21
13	XGBoost model 1 variable importance. . . . .	22
14	XGBoost model 2 variable importance . . . . .	22
15	LGBM model 1. The black series is the daily log returns . . . . .	23
16	LGBM model 1 variable importance . . . . .	24
17	Training and testing period used for hyper-parameter tuning. . . . .	24
18	LGBM model 2. The black series is the daily log returns . . . . .	25
19	LGBM model 2 variable importance . . . . .	25
20	LGBM model 3 variable importance . . . . .	26
21	CatBoost model. The black series is the daily log returns . . . . .	27
22	CatBoost model variable importance . . . . .	27
23	Neural network training period and validation period . . . . .	28
24	FFNN model. The black series is the daily log returns . . . . .	29
25	RNN model. The black series is the daily log returns . . . . .	30
26	LSTM model. The black series is the daily log returns . . . . .	31

---

---

## List of Tables

1	Descriptive statistics for daily EURUSD log-returns. Time period; 2009:01-2020:09. Source; Bloomberg. . . . .	4
2	QR-IM: ATM volatility, 25 delta risk reversal coefficients, along with the hit-percentage	18
3	Random forest performance . . . . .	20
4	Gradient Boost training set performance . . . . .	21
5	XGBoost model 1 performance . . . . .	21
6	XGBoost model 2 performance . . . . .	22
7	LGBM created data set performance . . . . .	23
8	LGBM model 1 performance . . . . .	23
9	LGBM model 2 performance . . . . .	25
10	LGBM model 3 performance . . . . .	26
11	CatBoost performance . . . . .	26
12	37% LGBM model 2 with 63% CatBoost, model performance . . . . .	27
13	FFNN performance . . . . .	28
14	RNN performance . . . . .	29
15	LSTM performance . . . . .	30
16	Summary of all model breach ratios . . . . .	32
17	Summary of all model if breach values, i.b. and the if not breach values, i.n.b . . .	33
18	Summary of all model containing p-values for the Christoffersen test and DQ-test (with four lags). . . . .	33

---

---

# 1 Introduction

The ability to model Value at Risk (VaR) with high accuracy is an important tool for quantifying risk in financial markets [1]. VaR is an estimate of the loss that will be exceeded with a small probability during a fixed holding period. Thus, offering traders, investors, and institutions information regarding risk estimators and potential investment insight. Besides a prominent role in regulatory frameworks, VaR will continue to be important for financial institutions as a measure of market risk [2].

This paper extends the work of de Lange et. al. [2]. As proposed by the authors, machine learning models could further improve the proposed model by introducing nonlinearity to the predictions. Therefore, the aim of this paper is employing different machine learning models and techniques in order to improve the predictions.

By conducting a literature study on machine learning methods we discovered that both neural networks and ensemble methods had been used for VaR predictions. Recurrent neural network and long short-term memory neural network, stood out among the neural network models. Both models yielding more accurate forecasting results for time series data compared to the feed forward networks, however, still widely used. Amidst the ensemble methods, the random forest model was used the most frequently. Gradient boosting methods have also been used, but less commonly. Ensemble methods and neural networks are state of the art models in the machine learning community today.

In this study both neural network models and ensemble models are going to be used for improving the predictions of VaR. Gradient boosting methods as well as random forest are going to be tested along side the recurrent neural network, long short-term memory neural network and feed forward networks. The various models are created using different model architectures as well as hyperparameter tuning, and model stacking for the ensemble models.

The methodology entails investigating several options for tuning and improving the two concepts of ensemble methods and neural networks. In addition, the training and validation data set are constructed using the QR-IM model. This model was proposed in the paper by de Lange et. al. [2] for forecasting the Value-at-Risk of foreign exchange markets. All models will be trained with the same data, and validated on the same out of sample period.

To the best of our knowledge, this paper contributes to the literature being the first to utilize information in the volatility surface - more precisely at-the-money volatility and risk reversals as stand-ins for higher order moments - combined with machine learning and quantile regression, to provide accurate VaR estimates.

Our main findings are:

1. The ensemble models achieves good estimates across all quantiles. However, two models stand out, the LightGBM model and the Categorical boost model, when comparing to the benchmark, QR-IM model. Both models yielding estimates at a better and equal performance to the benchmark model.
2. The neural network models are in general quite unstable and could benefit from more training data and perhaps a better model architecture.
3. Model stacking and hyperparameter tuning delivered an improved model when it comes to the overall performance of the predictions.

The remainder of the paper is organized as follows: Chapter 2 provides an overview of the literature, Chapter 3 presents the data, Chapter 4 describes the methodology, Chapter 5 lists the results and discussion, and Chapter 6 concludes.

---

## 2 Literature review

Quantile regression methods are often applied to Value at Risk forecasting. Engle and Manganelli [3] proposed the CAViaR method. This model can directly model the quantiles instead of modelling the whole distribution. Taylor [4] proposed the exponentially weighted quantile regression, or EWQR model in short, for estimating Value at Risk. He found that this model outperformed both GARCH-based methods and CAViaR models. Chen and Chen [5] found that the quantile regression approach outperforms the variance-covariance approach.

A few papers have applied quantile regression in the context of forecasting volatility or VaR of foreign exchange rates. Taylor [6] finds that a quantile regression approach delivers a better fit to multi-period data when forecasting volatility compared to variations of the GARCH(1,1) model. Huang et al. [7] use quantile regression to forecast foreign exchange rate volatility. Jeon & Taylor [8] include implied volatility as an additional regressor in CAViaR models and find increased precision of FX VaR estimates.

In order to make accurate forecasts of Value at Risk for foreign exchange rates, effective explanatory variables are essential. Chang et al. [9] provide an outline of different forecasting objectives using options data, including option-implied individual moments. Barone-Adesi et al. [10] and Huggenberger et alia [11] show how to use options data to compute forwardlooking VaR and Conditional-VaR measures. The information content of option combinations - such as the risk reversal - has been studied in the context of arbitrage-free option pricing and hedging both in papers by Bossen et al. [12] and Sarma et al. [13]. De Lange, Risstad and Westgaard [2] forecast the Value-at-Risk of foreign exchange markets using both at-the-money option contracts from the over-the-counter foreign exchange inter bank market to model volatility, and risk reversals as a proxy for higher order moments. The model proposed, the QR-IM model, outperformed ordinary base line models for VaR forecasts.

A number of previous studies has confirmed the forecasting ability of a plain vanilla Feedforward neural network over traditional statistical models. However, standard neural networks have limitations. Most notably, the model relies on the assumption of independent data observations, which presents a problem when data points are related through time. In order to overcome this problem, Bijelic and Ouiggane [14], use a Gated Recurrent Unit type of neural network to produce one-step-ahead volatility forecasts of the EURUSD exchange rate. Their model is outperformed by a GARCH(1,1) model for the VaR 95%. Xu et al. [15] modelled the Value-at-Risk of foreign exchange rates with three different neural network models; the deep belief network (DBN), multilayer perceptron (MLP), and long short-term memory network. Other studies that look into neural networks for predicting foreign exchange rates are: Yaya Heryadi and Antoni Wibowo [16] and He et al. [17].

Neural networks have been used in a vast variety of studies trying to model Value at Risk. Petneházi [18] used convolutional neural networks to forecast Value at Risk. Modifying the algorithm slightly, the convolutional networks can estimate arbitrary quantiles of the distribution, not only the mean, thus allowing the network to be applied to VaR-forecasting. Pradeepkumar and Ravi [19] forecasted financial times series volatility using a developed particle swarm optimization trained quantile regression neural networks, named SPOQRNN. They compare this model to three traditional forecasting models including GARCH, multi layer perceptron, general regression neural network, random forest and more. The SPOQRNN performed better than the rest.

A few authors have used Artificial Neural Network (ANN) for forecasting Value at Risk. Taylor [20] applied a quantile regression neural network approach to estimate the conditional density of multiperiod returns. The model is compared to GARCH-based quantile estimates on daily exchange rates. Xu et al. [21] developed a new quantile autoregression neural network (QARNN) model based on an artificial neural network architecture. By optimizing an approximate error function and standard gradient based optimization algorithms, QARNN outputs conditional quantile functions recursively. This allows the QARNN to explore non-linearity in financial time series. Yen et al. [22] used ANN to forecast VaR on a stock index. The authors stated that the model benefit from adding more exogenous parameters in the estimation process, such as interest rates.



---

Long short-term memory neural network has been used by Yan et al. [23] in a quantile regression framework to learn the tail behavior of financial asset returns. The model captures both the time-varying characteristic and the asymmetrical heavy-tail property of financial time series. The authors combine the sequential neural network with a self constructed parametric quantile function to represent the conditional distribution of asset returns. Kakade et al. [24] propose a hybrid model that combines LSTM and a bidirectional LSTM with GARCH, to forecast volatility. The model is evaluated on periods with extreme volatility: the 2007-09 Financial Crisis and the Covid Recession of 2020–21. The proposed model gives significant improvement in the quality and accuracy of the VaR forecasts compared to bench mark GARCH models.

Ensemble methods has been used by a number of studies estimating VaR. Andreani et al. [25] introduced the use of mixed-frequency variables in a quantile regression framework by merging the Quantile Regression Forest algorithm and the Mixed-Data-Sampling model. The empirical application of the model delivered adequate VaR forecasts and outperformed popular existing models used in VaR forecasting, such as the GARCH model, in terms of quantile loss. Jiang et al. [26] proposed a hybrid semi-parametric quantile regression random forest approach to evaluate Value at Risk. The model was used to explain the non-linear relationship in multi-period VaR measurement. Görden et al. [27] used a generalized random forest (GRF) for predicting Value-at-Risk for cryptocurrencies. They found that random forest outperformed quantile regression methods, including GARCH-type and CAViaR models, when tailored to conditional quantiles. The authors state that the adaptive non-linear form of GRF appears to capture time-variations of volatility and spike-behaviour in cryptocurrency return especially well in contrast to more conventional financial econometric methods. Gradient boosting methods were tested by Cai et. al. [28] for modeling VaR. They discovered the method to be effective at capturing risk.

---

## 3 Data

We apply our ensemble- and neural network models to empirical data for the EURUSD currency cross, taking implied volatility quotes as input data, and examine the models' predictive properties. Daily exchange rates and implied volatility quotes are sourced from Bloomberg, covering the period from 2009:01 to 2020:12.

### 3.1 Spot-rate returns

The daily returns are calculated as  $r_t = \ln(S_t/S_{t-1})$ , where  $S_t$  is the spot exchange rate at time  $t$ . In our case, this is the daily return of a dollar measured in euros. Table 1 displays descriptive statistics for daily log-returns from 2009:01 to 2020:12 and Figure 1 plots the time series correspondingly. The return series exhibit the stylized facts that have been widely documented in the financial economics literature; with unconditional means close to zero, clustering of volatility and fat-tailed return distributions.

n	2930
Mean	-0.0001
Std. dev	0.0053
Skewness	0.0330
Kurtosis	4.7508
Min	-0.0229
Max	0.0295

Table 1: Descriptive statistics for daily EURUSD log-returns. Time period; 2009:01-2020:09. Source; Bloomberg.

### 3.2 Investigating the data

Figure 2 displays time series for at-the-money volatilities (ATM) and 25-delta risk reversals (RR) for european EURUSD options with one week to expiry, and reveals the stochastic nature of the volatility surface. ATM levels have spiked around important economic events, such as the Brexit vote in June 2016 and the Covid 19 outbreak in March 2020. Panel (c) shows that the sign of the RR has changed over time and taken both positive and negative values, which in itself is an interesting observation. If the risk reversal reflects the relative probability of depreciation and appreciation of a currency, time-varying sign and magnitude of the risk reversal can be interpreted as an indication of time varying probability of tail events. Panels (b) and (d) display empirical distributions, which are skewed and leptokurtic for both variables. ATM volatility is naturally bounded below by zero, but spikes during periods of market turmoil and thus cause a heavy right tail. The risk reversal displays a highly non-normal, heavy-tailed empirical distribution.

### 3.3 Testing the data set

In order to test the data for stationarity we performed an augmented Dickey-Fuller unit root test and conclude that our data is stationary. Further to rule out multicollinearity, we ran a variance inflation factor test, or VIF in short. No sign of multicollinearity was discovered. We also tested our data for heteroskedasticity using the Breusch-Pagan test and serial correlation between the explanatory variables using a Breusch-Godfrey test. The Breusch-Pagan test indicated some sign of heteroskedasticity, whereas no sign of serial correlation was found.

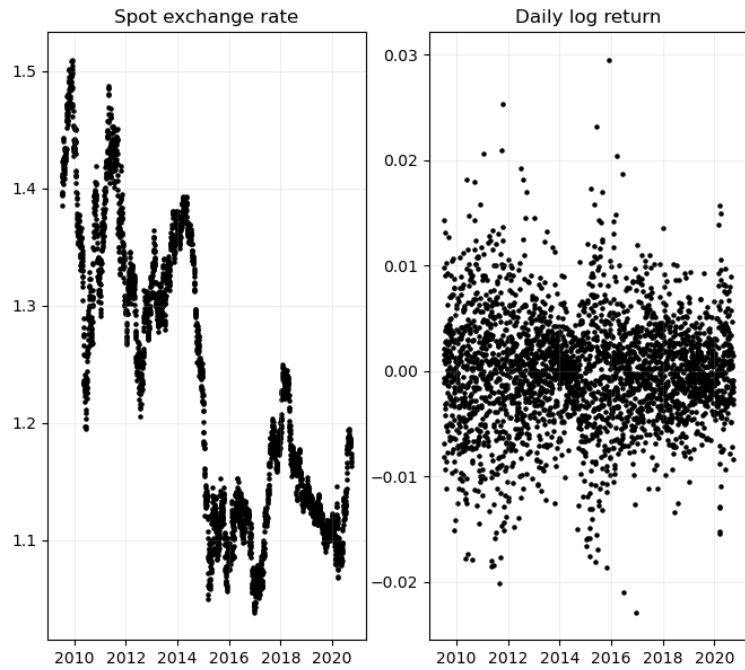


Figure 1: EURUSD spot exchange rates (left panel) and daily log-returns (right panel). Time period;2009:01-2020:09. Source; Bloomberg.

Figure 2 displays the time series and empirical distributions for implied moments of 1-week to expiry options; at-the-money volatilities [panels (a) and (b)] and 25 delta risk reversals [panels (c) and (d)]. Time period; 2009:01-2020:09. Source; Bloomberg.

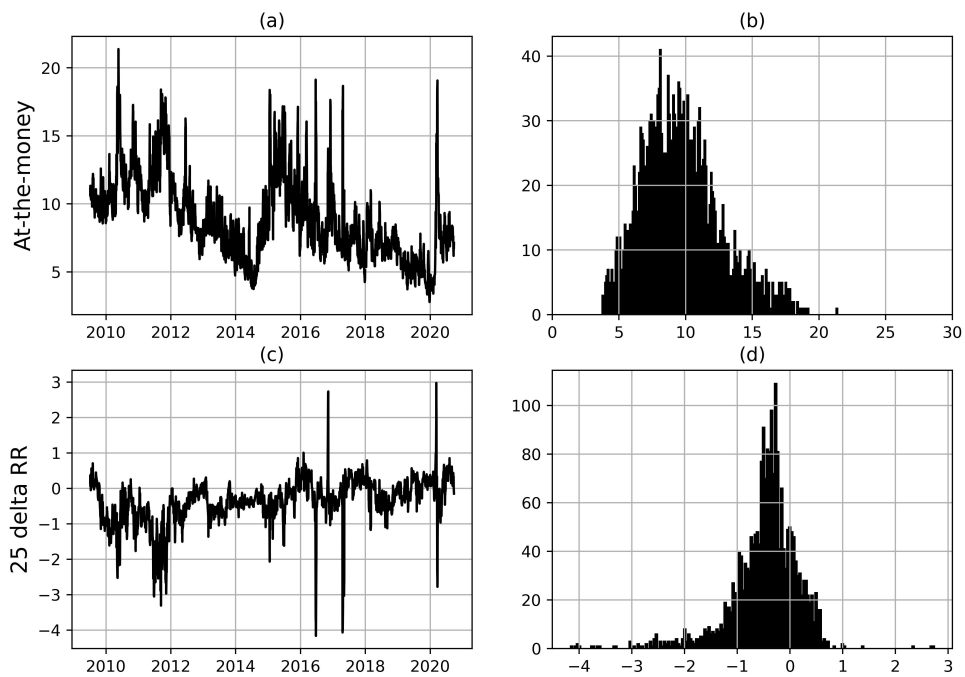


Figure 2: EURUSD spot exchange rates (left panel) and daily log-returns (right panel). Time period;2009:01-2020:09. Source; Bloomberg.

---

## 4 Methods

In this chapter, we provide a brief outline of the models we have used to produce our VaR forecast. We demonstrate how the data set is generated, briefly and generically explain the QR-IM model, ensemble methods and neural networks. We also provide a short note on preprocessing of data and implementation of models. A subsection is devoted to the specific ensemble models and neural networks, which we have implemented on our data.

### 4.1 Generating the training data

In order to examine the different machine learning methods, we need to create a proper data set containing the required quantiles. In this study we use two methods for creating the training data set. First, we employ the methods proposed by de Lange et al. [2] using the QR-IM model. Our second approach is using gradient boosting to generate quantiles based on the generated predictions. This is achieved by the light gradient boosting-machine model and the gradient boosting model. Both models have built-in options for quantile regression. The second method using QR-IM is more complicated, and is explained below.

#### 4.1.1 The quantile regression implied moments model

De Lange et al. [2] propose a quantile regression implied moments (QR-IM) model. We employ this model for generating benchmark, in sample quantiles for our currency data. The model is based on the hypothesis that the volatility surface of OTC FX options contains information, which can be utilized to improve the accuracy of VaR estimates. Similar to [2], we study data for at-the-money (ATM) options and risk reversals (RR).

*At-the-money options* are struck at the FX forward rate. ATM options have an initial delta of 50%. The ATM implied volatility is the risk-neutral expectation of spot rate volatility over the remaining life of the option.

*Risk reversals* involve the simultaneous sale of a put option and purchase of a call option. The two options are struck at the same delta. At the outset a 25% delta risk reversal will have a combined delta of 50% and very little sensitivity to gamma and vega because of the offsetting effect of the long and short position. Risk reversals are usually quoted as the difference in implied volatility of similar call and put options with the equal delta. The risk reversal reflects the difference in the demand for out-of-the money options at high strikes compared to low strikes. Thus, it can be interpreted as a market based measure of skewness; the most likely direction of the spot movement over the expiry period.

In the general case, the simple linear quantile regression model is given by:

$$Y_q = \alpha + \beta X + \epsilon_q \quad (1)$$

where the distribution of  $\epsilon_q$  is left unspecified. The expression for the conditional  $q$  quantile,  $0 < q < 1$ , is defined as any solution to the minimization problem:

$$\min_{\alpha, \beta} \sum_{t=1}^T (q - I_{Y_t \leq \alpha + \beta X_t})(Y_t - (\alpha + \beta X_t)) \quad (2)$$

where

$$I_{Y_t \leq \alpha + \beta X_t} = \begin{cases} 1 & \text{if } Y_t \leq \alpha + \beta X_t \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

In the QR-IM model, the conditional quantile function can be expressed as

---


$$\widehat{VaR}_{q,t+1} = \hat{\alpha}_q + \hat{\beta}_q^{ATM} ATM_t + \hat{\beta}^R R_q RR_t + \epsilon_{q,t} \quad (4)$$

A unique vector of regression parameters  $[\hat{\alpha}_q, \hat{\beta}_q^{ATM}, \hat{\beta}^R R_q]$  can be obtained for each quantile of interest, and the whole return distribution can be found, given observed values for the at-the-money volatility (ATM) and the risk reversal (RR).

## 4.2 Value-at-Risk

VaR is a statistical risk measure of potential losses and summarizes in a single number the worst loss over a target horizon that will not be exceeded with a given level of confidence. Despite several other competing risk measures proposed in the literature, VaR has effectively become a cornerstone of internal risk management systems in financial institutions, following the success of the JP. Morgan RiskMetrics system and nowadays form the basis of the determination of market risk capital.

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. Formally, the VaR at a level  $\alpha$  of the profit and loss distribution  $X$  is defines as:

$$VaR_\alpha = \min\{m : P(L \leq m) \geq 1 - \alpha\} \quad (5)$$

Using (5), one can therefore interpret  $VaR_\alpha(x)$  as the smallest  $m$  such that the probability of the loss being at most  $m$  is at least  $1 - \alpha$ . For example, 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 parametric and nonparametric approaches. For the former, a normal distribution is often assumed and  $\mu$  and  $\sigma^2$  is the mean and variance of the investment based on a sample of previous returns. This is called the mean-variance approach. In the nonparametric 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.

**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 common assumption for financial data is that this distribution, based on historical data (observations of  $R_t$  for  $t < T$ ), tends to be a Gaussian distribution with some mean  $\mu$  and variance  $\sigma^2$ . It is thus easy to compute the  $1 - \alpha$ -quantile and estimate the  $VaR_\alpha(x)$  amount from the quantile using the resulting distribution.

Instead of employing the complete historical window up to the present time  $t = 0$ , a so-called lookback period is selected, which is represented by the symbol  $d$ . The mean and standard deviation for the lookback period are then computed. Long lookback periods are recommended since there is likely to be a significant difference between the means and variances of different eras. This approach entails that the VaR is not calculated for the first  $d$  days. Thereafter, for each  $d$ -day period, one calculates:

$$\hat{\mu}_t = \frac{1}{d} \sum_{j=0}^{d-1} R_{t-d+j}, \text{ for } t = 1, \dots, T \quad (6)$$

The variance is calculated as:

$$\hat{\sigma}_t^2 = \frac{\sum_{j=0}^{d-1} (R_{t-d+j} - \hat{\mu}_t)^2}{d - 1}, \text{ for } t = 1, \dots, T \quad (7)$$

---

The standard deviation is of course the square root of the expression (7),  $\hat{\sigma}_t$ , is frequently referred to as the volatility. Both (6) and (7) are moving averages. For a day  $t$  in  $[1, T]$ , the prediction for VaR at a confidence level  $\alpha$  is calculated by taking the  $1 - \alpha$ -quantile of the distribution  $\mathcal{N}(\hat{\mu}_t, \hat{\sigma}_t)$ .

**Historical Simulation:** In principle, historical simulation operates similarly to the mean-variance approach, i.e., by picking a lookback period and calculating the  $1 - \alpha$ -quantile. However, in historical simulation, this is accomplished by simply sorting the sample returns, followed by selecting the quantile. In a sample of independent copies  $R_1, \dots, R_n$  of the return distribution  $R$  for an investment  $X$ , the empirical estimate of  $VaR_\alpha(x)$  is given by

$$\widehat{VaR}_\alpha(x) = R_{[n\alpha]+1, n} \quad (8)$$

where  $R_{1, n} \geq \dots \geq R_{n, n}$  are the sample returns ordered by ascending size, and  $[x] = \max\{m \in \mathbb{Z} \mid x \leq m\}$ , also known as the floor function. To highlight this with an example, suppose that there are  $n = 220$  samples of the return  $R$  and one seeks the VaR at confidence  $\alpha = 0.01$ . This would require ordering the samples such that  $R_{1, 220} \geq \dots \geq R_{220, 220}$  and choosing  $\widehat{VaR}_{0.01}$  as  $R_{[220 \cdot 0.01]+1, 220} = R_{[2.2]+1, 220} = R_{3, 220}$ .

### 4.3 Ensemble learning

In this study we employ two base methods for estimating VaR, ensemble learning and neural networks. Ensemble learning is the process of using multiple models, often called weak learners, trained over the same data and combined to obtain better results. Weak learners, or base models, are models that on average perform slightly better than random chance. This is either because they have a high bias or because they have too much variance to be robust. The idea of ensemble methods is to attempt reducing bias and/or variance of such weak learners by combining several of them, in order to create a strong learner that achieves better performance.

Low bias and low variance are two of the most desirable features of a model. There is also a trade off between degrees of freedom and the variance of a model. Introducing too many degrees of freedom can cause high variance. Thus, a limit of the amount of degrees of freedom is required to avoid high variance and to be more robust. This is known as bias-variance trade off.

When combining weak learners we need to assure that our choice of weak learners is consistent with the way we aggregate the models. If we choose base models with low bias but high variance, we should employ an aggregating method that tends to reduce variance, whereas if we choose base models with low variance but high bias, we should utilize an aggregating method that tends to reduce bias. In general there are three major kinds of algorithms that aim at combining weak learners: bagging, boosting and stacking.

Bagging learns homogeneous weak learners independently and combines them following some kind of deterministic averaging process. Boosting, which also often considers homogeneous weak learners, learns them sequentially and combines them following a deterministic strategy. Stacking, learns weak learners in parallel and combines them by training a meta-model which outputs a prediction based on the different weak model predictions. In general, both boosting and stacking mainly try to produce strong models with less bias than their components, whereas bagging mainly focuses on getting an ensemble model with less variance than its components.

### 4.4 Deep learning

This section gives a short introduction to deep learning and its concepts. Deep learning refers to the machine learning methods using deep neural networks to approximate some unknown function based only on the inputs and expected outputs. The section starts by explaining the concepts of artificial neural networks before mentioning some activation functions relevant to this study. Furthermore, the learning process of the neural network will be explained along with optimizers and the concept of batch and epoch.

---

#### 4.4.1 Artificial neural network

Artificial neural networks (ANNs), hereafter referred to as neural networks (NNs), recognize underlying relationships in a data set through a process that mimics the way the human brain operates. NNs are non-linear functional approximators that can be tuned to approximate an unknown function based on observations and target outputs. NNs are structured in consecutive layers. Each layer takes an input, applies a transformation, and returns an output. The perception, originally described by Rosenblatt in 1958 [29], acts as the inspiration for the neuron, the foundational building piece of the layer. There is an associated weight for each element in the input vector to the neuron. The bias of the neuron can optionally be added to the input. The neuron then determines an output by adding the inputs and the weights assigned to each input:

$$y = \sum_i^n x_i w_i + b \quad (9)$$

An arbitrary positive number of neurons that each provide an output can make up a layer in a NN. Input can be a vector of size  $m$  and output a vector of size  $n$ , equal to the number of neurons in the layer. Calculating the output vector involves multiplying the input vector by the weight matrix of the layer:

$$\mathbf{y} = \mathbf{x}\mathbf{W} + \mathbf{b} \quad (10)$$

Where  $\mathbf{x}$  is the input vector,  $\mathbf{y}$  is the output vector,  $\mathbf{W}$  is a  $m \times n$  matrix containing the weights for the  $n$  neurons in the layer and  $\mathbf{b}$  is the vector of biases.

#### 4.4.2 Activation functions

As mentioned in Section 4.4.1, neural networks are organized in successive layers, each layer computes an output given an input as described in Equation (10). The output of one layer is then subjected to a nonlinear transformation before being passed as input to the following layer. This is referred to as an activation function. Below is a list of commonly used activation functions:

##### The logistic function

The logistic function, also known as the sigmoid activation function, transforms the values into the range  $[0, 1]$ :

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (11)$$

##### Tanh

The hyperbolic tangent function, simply referred to as tanh, is similar to the sigmoid activation function, but instead outputs values in the range  $[-1, 1]$ :

$$\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (12)$$

##### ReLU

The Rectified Linear Unit (ReLU) activation function outputs values in the range  $[0, 1]$ . It does this by taking the maximum of the input and zero

$$\sigma(x) = \max(0, x) \quad (13)$$

All mentioned activation functions are displayed in Figure 3, in blue, with their derivatives in yellow.

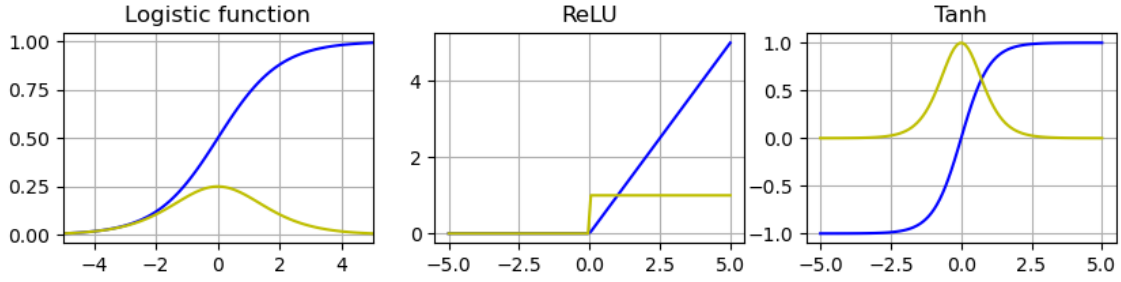


Figure 3: Activation functions

#### 4.4.3 Training a neural network

The process of adjusting the layer weights until the network satisfactorily approximates the target function is known as training the neural network. The function that the neural network is attempting to approximate is denoted by  $F$ . An objective function, frequently referred to as the loss function, is used to measure how closely the network approximates  $F$ . When the neural network accurately approximates  $F$ , the loss function has a global minimum. It calculates the distance between the output  $\hat{\mathbf{y}}$  to the target output  $\mathbf{y}$ .

One of the most commonly used loss functions is the mean squared error (MSE). It is defined as:

$$MSE = \frac{1}{n} \sum_{i=1}^n (\mathbf{y}_i - \hat{\mathbf{y}}_i)^2 \quad (14)$$

The neural network is trained by minimizing the loss function with regard to the neural network weights. The optimization problem is solved by calculating the partial derivatives of the loss function with regard to the weights. Let  $\frac{\partial L}{\partial w_i}$  stand for the partial derivative and  $L$  for the loss function. The process of updating the weights and biases is shown in Equation (15) and (16).

$$w_i = w_i - \eta \frac{1}{m} \sum \frac{\partial L}{\partial w_i} \quad (15)$$

$$b_i = b_i - \eta \frac{1}{m} \sum \frac{\partial L}{\partial b_i} \quad (16)$$

The learning rate of the neural network is denoted by the symbol **eta**. It is a small positive value that determines how large the steps along the gradient should be at each iteration. The term  $\frac{1}{m} \sum^m$  comes from averaging all gradients for all samples. This implies that 10,000 gradients would be calculated for each weight given a set of 10,000 samples of input and targets. Rumelhart [30] provides a thorough explanation of this procedure, which is known as backpropagation.

One thing to consider when modelling a deep neural network is that learning is carried forth with the derivatives. As seen in Figure 3, the derivatives of the logistic function and the tanh function quickly approaches zero. This means that the information is lost in a deep net. That is why the ReLU activation function is preferred in deep neural nets as the derivative is either 0 or 1.

#### 4.4.4 Optimizers

One of the many approaches for updating the weights of a neural network is traditional gradient descent. Here the derivatives (gradient) is minimized until finding a minimum. The method is advantageous in that it is simple to use and will yield the global optimum if the objective function is convex. One significant disadvantage is that the procedure may reach a local optimum and hence



---

come to an end when looking for the global optimum. Additionally, the computational complexity will be considerable for each iteration and possibly infeasible for huge data sets. This is because every iteration uses every piece of training data. The computational complexity will be  $\mathcal{O}(ND)$  for each iteration if we use  $N$  for the sample count and  $D$  for the data set dimension. To reduce this problem and the risk of hitting a local minimum, the *stochastic gradient descent* technique was created [31].

### Stochastic gradient descent

Here, instead of computing the precise value of the gradient using the whole data set, one sample from a small batch of samples is randomly selected to update the gradient during each cycle. Thus, the costs associated with calculating the new weights are drastically reduced because they are generated for one sample rather than the full data set. This approach computes the gradients averaged only on a randomly chosen subset  $S$  from the set of inputs and targets, also referred to as a minibatch. Goodfellow [32] claims that the predicted value of the gradient should then approximate the gradient determined using all samples.

Since only one sample is used, the gradient descent method does not uniformly converge in one direction because of the extra noise this causes. Furthermore, it is troublesome to have a single learning rate for all features, as rarely occurring features should ideally receive greater gradient updates and vice versa. The optimizer "adaptive moment estimation," or Adam for short, uses variable learning rates to address this problem. Adam has a high computational efficiency and reduces the requirement for earlier learning rate tuning [33].

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

$$g^{(\tau)} = \beta_1 g^{(\tau-1)} + (1 - \beta_1) \nabla \mathcal{L}(w^\tau)$$

$$V^{(\tau)} = \sqrt{\beta_2 V^{(\tau)} + (1 - \beta_2) (\nabla \mathcal{L}(w^\tau))^2} \quad (17)$$

With  $\beta_1$  and  $\beta_2$  being some exponential decay rates, resulting in the weight update equations

$$w^{(\tau+1)} = w^\tau + \Delta w^{(\tau)}, \quad (18)$$

$$\text{with } \Delta w^{(\tau)} = g^{(\tau)} - \xi * \frac{\sqrt{1 - \beta_2}}{1 - \beta_1} \frac{g^{(\tau)}}{V^{(\tau)} + \epsilon}$$

where  $\epsilon$  is set to a very small value to ensure that there is no division by 0.

#### 4.4.5 Batch size and epochs

The neural network performs 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  $b$ . Weight updates are performed for each batch, and the outcome is displayed in the value of the loss function of the network. When all batches have been propagated through the network, the training has been performed over 1 epoch. In other words, the number of epochs represents how many times all batches are transmitted over the network.

Naturally, the batch size selection has consequences because it impacts the gradient descent. Three different types of gradient descent are typically introduced depending on the batch size: stochastic gradient descent, where  $b = 1$ , mini-batch gradient descent, where  $1 < b < N$ , and batch gradient descent, where  $b = N$ . Since the weights are adjusted after each propagation, having a smaller batch size may speed up computation, but it may also result in less accuracy in the gradient value [32]. When compared to the batch approach, the gradient fluctuates more with the stochastic approach. A mini batch gradient descent method thus acts as a compromise between the two.

---

## 4.5 Pre-processing

When using different machine learning algorithms such as neural networks and ensemble methods, it is often imperative to scale the data in order to speed up the learning process and achieve faster convergence. This is typically accomplished by scaling the data to obtain a mean that is almost zero [34]. This step is especially important when working with tasks that involve a variety of features at various scales, such as estimating the likelihood of a person having between 0 and 5 children given their annual income (tens of thousands of dollars), height (roughly between 150 and 200 cm), and number of years of post-secondary education (usually between 0-6). Given the fact that log returns are already adequately centered around a mean of zero, the data at hand is essentially directly compatible with ensemble learning and a neural network [35].

## 4.6 Implementation

To construct ensemble methods different open source python packages has been used. From Scikit-learn the methods for random forest [36], gradient boost [37] and adaptive boosting [38], Microsoft introduces the light gradient-boosting machine [39]. From Distributed Machine Learning Community (dmlc) we get the extreme gradient boosting [40] and at last we obtain the categorical boosting [41] from Yandex.

We use Keras [42] to construct the neural network, 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, recurrent-layers and LSTM-layers, in addition to a multitude of different machine learning methods.

## 4.7 The chosen ensemble models

According to the literature review in Chapter 2, the random forest algorithm was more frequently employed to estimate VaR compared to the gradient boosting methods. Therefore, in this study different gradient boosting methods, as well as the random forest, is going to be tested for predicting VaR.

### Random forest

The random forest model is an ensemble of many weak learners, in this case decision trees. It can be applied to classification and regression problems. The regression procedure using random forest starts by splitting of features and then creates decision trees. Every tree makes its individual decision based on the data. The average value predictions from all trees become the final prediction.

### Gradient boosting

Gradient boosting on decision trees is a form of machine learning that works by progressively training more complex models to maximize the accuracy of predictions. Gradient boosting is particularly useful for predictive models that analyze ordered (continuous) data and categorical data. Gradient boosting benefits from training on huge data sets. Gradient boosting is one of the most efficient ways to build ensemble models. The combination of gradient boosting with decision trees provides state-of-the-art results in many applications with structured data. Gradient boost algorithms are build in an iterative way. First, the algorithm learns from the first tree to reduce the training error. The algorithm, then repeats this procedure until it reach a decent quality. In general it is not a good idea to build very big trees in boosting, since they over fit the data.

### Extreme gradient boosting

Extreme gradient boosting (XGBoost), is an improved version of the gradient boosting algorithm. In this algorithm, decision trees are created sequentially. Different weights are assigned to all the independent variables, which are then fed into the decision tree that predicts results. The weight of wrongly predicted variables by the tree is increased and the variables are then fed to the second decision tree. This is known as an greedy algorithm. These individual classifiers/predictors then ensemble to give a strong and more precise model. This works for regression, classification, ranking, and user-defined prediction problems.

---

### Light gradient boosting-machine

Light gradient boosting-machine, or known as LightGBM is a gradient lifting framework which is based on the decision tree algorithm. It can be used in classification, regression, and many more machine learning tasks. Compared to XGBoost it splits leaf-wise and chooses the maximum delta value to grow, rather than level-wise. LightGBM has a great advantage when performing hyperparameter optimization. This is because the different input-parameters are easily controlled.

### Category boosting

Categorical boosting, in short CatBoost, provides a way of performing classifications and rankings of data by using a collection of decision-making mechanisms. Results generated by the learners are weighted and classified based on the strengths and weaknesses of each learner.

## 4.8 The considered neural networks

There are plenty of different types of neural networks, each being more suitable depending on the task at hand. The recurrent neural network (RNN), which has a temporal component and can thus capture dynamic and time-dependent characteristics in the data, is frequently employed for time series. The long short-term memory (LSTM), a specific kind of RNN, has the ability to combine knowledge from the distant past with information from the present, depending on how the model is configured. In this study both the recurrent neural network and the long short term memory are chosen as models, in addition to the feed forward neural network.

### Feed forward neural network

The feed forward neural network (FFNN) is an artificial neural network. Here connections between the nodes do not form any type of cycle. The information moves only in one direction, forward in the network from the input nodes. The information is passed through the hidden nodes to the output nodes. Feed Forward Neural Networks with a single hidden layer is the most widely used neural network for forecasting [43].

### Recurrent neural network

A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes can create a cycle. Thus allowing output from some nodes to affect subsequent input to the same nodes. This allows it to exhibit temporal dynamic behavior.

### Long short term memory

The long short-term memory (LSTM) can consolidate information from far in the past with that which is more recent. A forget gate, an input gate, and an output gate make up the LSTM-cell. While the gates govern the information flow in and out of the cell, the fundamental function of the LSTM-cell is to recall information over time intervals. The gates essentially decide which information is remembered and which is forgotten. The information that the LSTM-cell remembers is kept in its cell state  $C_t$  and computed based on the input  $x_t$ . The results of the previous output  $h_{t-1}$  is kept in the forget gate.

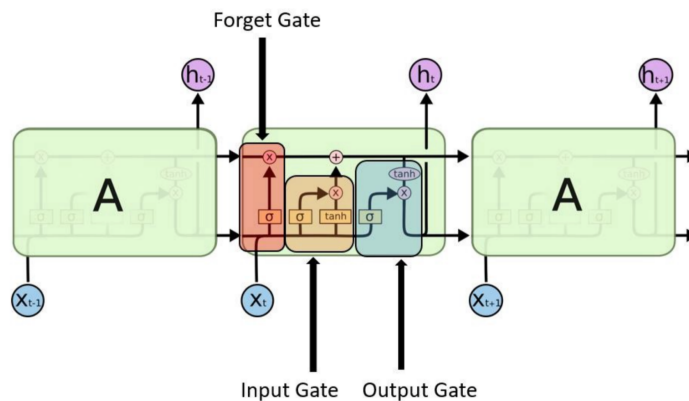


Figure 4: A long short term memory layer.

---

## 4.9 Hyperparameter architecture

The hyperparameter tuning is a little different for the ensemble methods and neural networks. Nevertheless, the general aim is to minimize the loss function and maximize accuracy while reducing bias and overfitting. For both the ensemble methods and the neural networks, choosing the correct architecture is non-trivial. When it comes to the ensemble methods, the chosen architecture is mostly based on the method in use, whether this is ordinary random forest or some gradient boost technique. Choosing a method also restricts a lot of the possible tuning. In this study, the process of hyperparameter tuning for the ensemble methods has been carried out by the use of software (if it exists), and trial and error. Typically the tuning parameters are the number of leaves, number of trees and the learning rate.

In a neural network there are a lot of possible variations. The hyperparameters that need to be adjusted in the neural network are the number of layers, number of nodes in each layer, the look-back period, activation functions, learning rate, batch size and number of epochs. This has been achieved through an iterative approach, optimizing one parameter at a time.

Another possibility for tuning the models is the training and validation period. The training data consists of data from 2009-07-07 to 2017-12-29. For the neural network there is also the training period. We have split the training period 80/20 percent, into the neural network training set and the test set respectively. The validation set is equal for both periods and consists of data from 2018-01-03 to 2020-09-25. Too short a training set can make the data prone to outliers, however it may be less overfitted. Too long a training period makes the data prone to overfitting, and the model may be useless for its actual task. This is because one cannot test the model on enough data to detect anomalies in the model.

### 4.9.1 Neural network hyper parameters

#### Interaction parameter

A neural network needs many input parameters in order to find any type of patterns in the data. In our data there are only two parameters. That may be too few for a neural network to work properly. Thus, a third parameter has been created. This is an interaction term between the two already existing parameters.

Interaction term;  $x_3 = \text{ATM volatility} \cdot 25\% \text{ delta risk reversal}$

#### Look back period

Due to the mechanics of LSTM-networks, the importance of specifying an optimal lookback period, 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 10, meaning that any given prediction used the past 10 days of data. The same look back period is used for the Recurrent Neural Network

#### Layers

A typical neural network has one input layer, one or more hidden layer(s), and one output layer. The *dense layer* is the most prevalent type of layer and, as its name suggests, is completely connected to neurons in other layers. The dense layer, which offers learning features for all combinations from the preceding layer, is quite simple in comparison to the LSTM and the RNN. The dense layer does not account for temporal aspects and memory as the LSTM and RNN-layer does. In this study, the FFNN-model, is constructed using dense layers only, as this is a simple feed forward neural network. However, the LSTM and the RNN-models need their respective LSTM and RRN layer, in addition to possible dense layers. For all three models the output layer consists of six nodes, one for each quantile. Stacking of multiple layers enables the network to construct deeper features from which the output can be based on [44]. However, a sparser model with fewer layers can outperform their denser counterparts in the context for financial forecasting. This stems from the low signal to noise ratio in financial data [45].

---

## Nodes

In general it is better to have too many nodes in a layer than too few [44]. This is because too few neurons might leave out important non linear connections in the data. Boyd and Kaastra, give different examples of potential rules of thumb for choosing the neurons in hidden layers. These rules are based on  $n$  input neurons and  $m$  output neurons. Here are three:  $\sqrt{n \cdot m}$  (Masters, 1993),  $0.75 \cdot n$  (Baily and Thompson, 1990), and between one half to three times  $n$  (Katz, 1992) [46].

## Miscellaneous improvements

The use of callbacks is a fundamental component in formulating the architecture of the neural network. A callback is a function that is used at specific points throughout the neural network's training [47]. In this study, callbacks served three functions: automatic adjustment of the learning rate, early stopping and model checkpoints.

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. The learning rate reduction factor, the number of epochs before adjustment (i.e., patience), and the variable that is monitored, can all be manually changed in this callback.

The callback *EarlyStopping* was used to introduce early stopping. This callback keeps track of the validation loss and stops the training of the neural network if the validation loss does not improve after 60 epochs. The variables in this callback can be manually changed, similar to *ReduceLRonPlateau*.

Lastly, the callback *ModelCheckpoint* was used to save the best model during training. Early stopping with a patience  $\geq 1$  has the risk of making the model worse after each epoch due to suboptimal weight updates. Thus, it is important that the model that produced the best loss be the one that is kept and used for prediction. In doing so, the model weights for that epoch are saved when the validation loss value has improved throughout training and later rewritten if the loss has improved at a later point in the training. The best model, which was saved while training, is loaded and utilized for prediction when the training is finished.

The model also employs dropout layers, which are positioned between other model layers. The layer sets a portion of the input units to zero at each training step, corresponding to a certain frequency in the range  $[0, 1]$ . This means that during network training, some weights are not updated. Therefore, the dropout layers aid in avoiding overfitting.

## 4.10 Explainable AI

We introduce Shapley values to enhance comprehension of how the machine learning models operate. Shapley (Shap) values arrive from Game Theory, and are frequently used for post-hoc model explainability in machine learning models. The idea is to see the impact each feature has on the target. For further explanation see the original paper [48].

## 4.11 Developed neural network models

For all models, a batch size of 5 has been used. Each model was allowed to run for 120 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.

### 4.11.1 Feed forward neural network

The feed forward neural network has three dense layers consisting of 12 nodes, and a dropout frequency of 0.2. After the three dense layers comes a suppress layer, with a dropout frequency of 0.2. This layer suppresses the data, and thus from the model developed, make the predictions more stable. All layers have a ReLU activation function. The model output are the predicted quantiles:  $\alpha : [0.01, 0.025, 0.05, 0.95, 0.975, 0.99]$ .

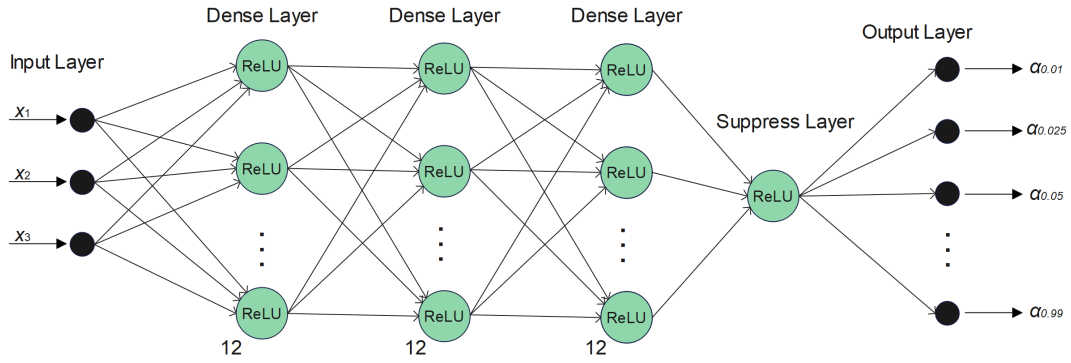


Figure 5: The feed forward neural network used in this thesis.

#### 4.11.2 Recurrent neural network

The recurrent neural network has two dense layers consisting of 12 nodes and between them a recurrent layer consisting of 8 nodes. All layers have dropout frequencies of 0.2 and a ReLU activation function. The model output is the predicted quantiles:  $\alpha : [0.01, 0.025, 0.05, 0.95, 0.975, 0.99]$ .

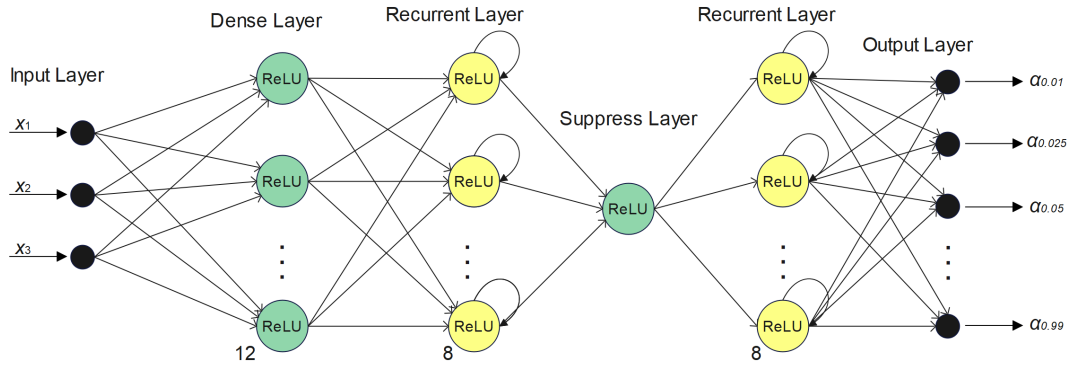


Figure 6: The recurrent neural network used in this thesis.

#### 4.11.3 Long short term memory neural network

The LSTM neural network starts with a dense layer of 12 nodes. Then comes two LSTM layers with 8 nodes, and in between them a suppress layer consisting of one node. Also, here the model needed a suppress layer to make the model more stable. All layers have dropout frequencies of 0.2 and a ReLU activation function. The model output is the predicted quantiles:  $\alpha : [0.01, 0.025, 0.05, 0.95, 0.975, 0.99]$ .

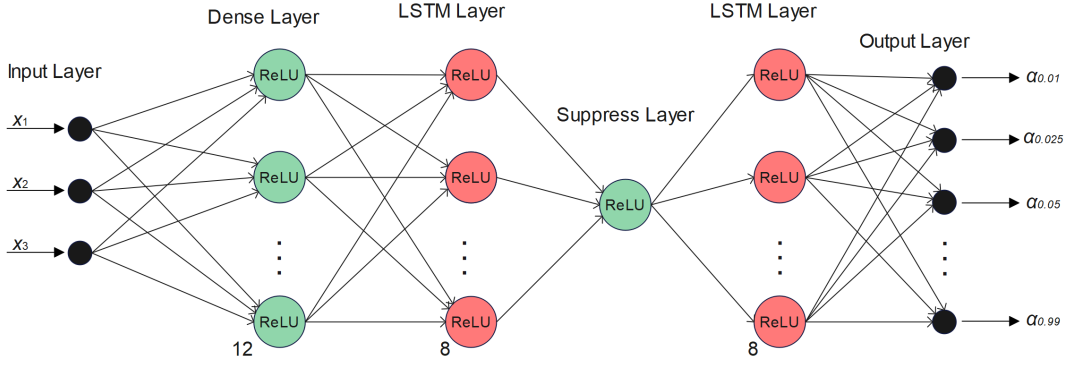


Figure 7: The LSTM neural network used in this thesis.

## 4.12 Evaluating the models

We let  $I(A)$  denote the indicator function, i.e., a function which returns 1 if the event  $A$  occurs and 0 if not, represented by (19). The true return is given by  $R$  while the predicted return (VaR-estimate) is given by  $\hat{R}$ .  $N$  provides the total number of predictions. We introduce the following metrics with this notation, which are used to examine superiority among the VaR models:

$$I_n(A) = \begin{cases} 1 & \text{if the event } A \text{ occurs.} \\ 0 & \text{else} \end{cases} \quad (19)$$

If the VaR is specified at confidence level  $\alpha$  the breach level should roughly be equivalent to  $\alpha$ . The *Breach ratio* is given as:

$$\text{Breach Ratio} = 100 \cdot \frac{\sum_{i=1}^N I(\hat{R}_i < R_i)}{N} (\%) \quad (20)$$

Calculating the nominal breach of the model over the total number of predictions is one method of evaluating performance. The model with smaller *Sum if breach* is favored given models with comparable breach ratios since this would suggest that the model is at least closer to the true value of the loss. The *Sum if breach* is given as:

$$\text{Sum if breach} = \sum_{i=1}^N I(\hat{R}_i < R_i)(R_i - \hat{R}_i) \quad (21)$$

It is also interesting to know how close a prediction is to the predicted medium when no breach occurs. Thus, the *Sum if no breach* is given as:

$$\text{Sum if no breach} = \sum_{i=1}^N I(R_i < \hat{R}_i)(\hat{R}_i - R_i) \quad (22)$$

Both formulas are expressed for the higher quantiles. The same formulas, however, can be used for the lower quantiles.

In addition, the following 4 metrics are used for further comparison:

$$\begin{aligned} \text{Total sum if breach} &= \sum_{q=\alpha} |\text{Sum if breach}_\alpha| \\ \text{Total sum if no breach} &= \sum_{q=\alpha} |\text{Sum if no breach}_\alpha| \\ \text{Max. VaR} &= \max(\hat{R}_1, \dots, \hat{R}_N) \\ \text{Min. VaR} &= \min(\hat{R}_1, \dots, \hat{R}_N) \end{aligned}$$

## 5 Results

In this section we present results. First, we present the proposed QR-IM quantile regression model and the estimated coefficients. Second, the different ensemble method models are presented along with the estimation results. Third, the neural networks are presented along with the estimation results. Lastly, a summary of all the models are presented and discussed. Throughout all estimates, a fixed training window from July 1, 2009 to December 31, 2017 is used, and the out-of-sample period covers January 1, 2018 to September 28, 2020.

### 5.1 Estimating the quantile regression VaR model

Table 2 displays the estimated quantile regression coefficients for the ATM variables (middle panel) and the 25 delta RR variables (bottom panel). The model is estimated on daily Bloomberg data during the in-sample period (2009:07-2017:12). The coefficients are scaled by 100 for readability. As for the sign, magnitude, shape, and statistical significance of regression coefficients, the results are consistent for both the ATM variable and the RR variable.

quantile - $\alpha$	1.0 %	2.5 %	5.0 %	95.0 %	97.5 %	99.0 %
Constant	-0.61 (i)	-0.21	-0.07	0.09	0.05	0.12
ATM	-0.06 (i)	-0.09 (i)	-0.08 (i)	0.09 (i)	0.12 (i)	0.15 (i)
25-delta RR	0.21 (i)	0.16 (i)	0.10 (ii)	0.16 (i)	0.30 (i)	0.38 (i)
<i>Breach Ratio</i>	1.04	2.48	4.97	94.99	97.43	99.01

Table 2: QR-IM: ATM volatility, 25 delta risk reversal coefficients, along with the hit-percentage

The breach ratios of the estimated quantiles of the in-sample data is also shown in Table 2. The breach ratio displays good coverage throughout the data with respect to the quantiles estimated. This is in line with the coefficients being significant.

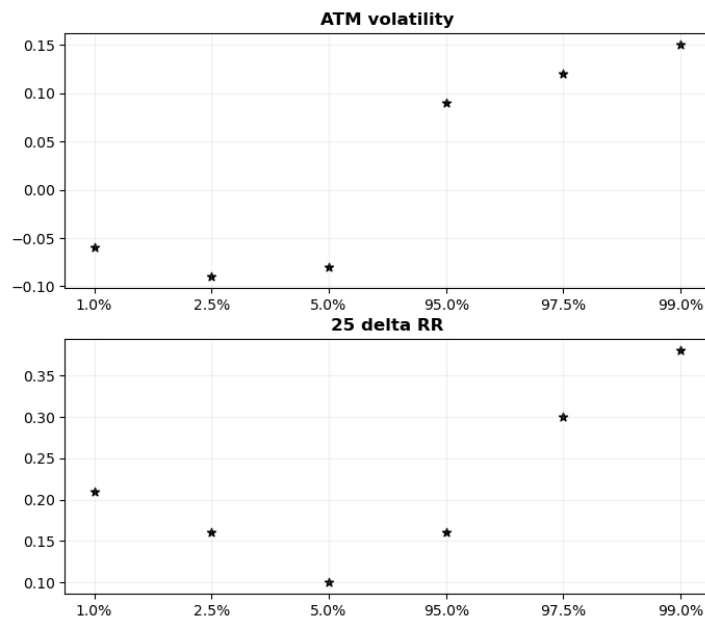


Figure 8: Quantile regression coefficients QR-IM

The estimated ATM coefficients support the view that the ATM is an indicator of overall market risk. The coefficients show a nonlinear, increasing relationship between the quantiles and the ATM regression coefficients. The coefficients are projecting the shape of the tails of the conditional



return distribution. This arises from the coefficients being negative for the lower quantiles and positive for the higher quantiles. In addition, the absolute value of ATM coefficients are slightly higher in the right tail compared to the left tail. This indicates a non-linear relationship between implied volatility and returns. Figure 8 shows the coefficients.

Figure 8 reveals that the RR coefficients form a U-shaped pattern, and are all positive. This indicates that implied volatility is more important when estimating both ends of the tails. Normally, for VaR estimates, the 1% and 99% -quantiles are the most valued. This is because they estimate the outlines of the VaR. The implied volatility is thus an great asset to the model. However, because all coefficients are statistically significant, all quantile-estimates of the return will improve when including the implied volatility.

## 5.2 Ensemble methods

We now present results for the ensemble methods. We have tested different uses of the algorithms for Random Forest, Extreme Gradient Boosting, Light Gradient Boosting, and Adaboost.

The data has been split in two for training and validation. The first part starts at the beginning of the sample period and lasts until the end of 2017. This part of the data is used for training the model. The second part of the data starts in 2018 and last until the end of the data-period. This part is used for validation and testing the performance of the model. Both periods are illustrated in Figure 9.

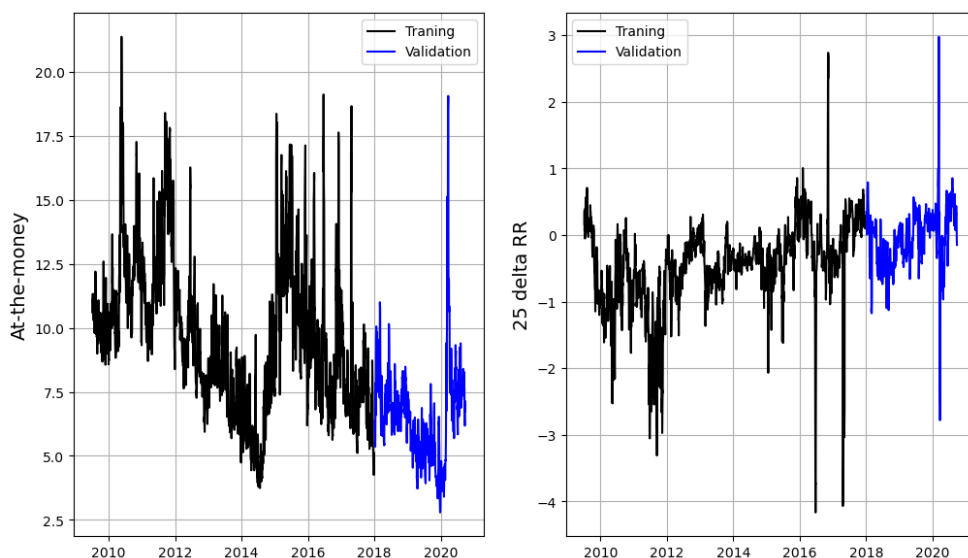


Figure 9: Ensemble training period and validation period

### 5.2.1 Random Forest

In this section we describe how the random forest algorithm has been tested. The data used in the training originates from the data generated by the QR-IM model in Equation (4), discussed in the previous section. The model is trained individually on each quantile. The in sample data, for RR and ATM volatility, along with the in sample estimated quantiles in Section 5.1 are used for training the model. For testing, only the sampling data for RR and ATM volatility is used. The results are listed in Table 3.

The first thing to notice is the increased performance for estimates for higher-order quantiles; 95%, 97.5% and 99%. Second, *the sum if no breach* increases as one moves further away from the mean. This makes sense as these estimates should be further away from the sampling data compared to quantiles closer to the mean. Third, as expected, *the sum if breach* is less further into the tails of

the distribution. Fourth, *the sum if breach* is almost equal to zero for the quantiles farthest out in the tails. This signals that the model yields promising results when predicting spike behavior.

quantile - $\alpha$	1.0 %	2.5 %	5.0 %	95.0 %	97.5 %	99.0 %
<i>Breach Ratio</i>	0.7	2.1	5.75	95.09	97.62	99.44
<i>Sum if breach</i>	-0.005	-0.018	-0.054	0.067	0.022	0.005
<i>Sum if no breach</i>	-7.421	-5.641	-4.483	4.729	5.982	7.650
<i>Min. VaR</i>	-0.023	-0.022	-0.019	0.004	0.005	0.006
<i>Max. VaR</i>	-0.008	-0.005	-0.004	0.016	0.023	0.029

Table 3: Random forest performance

The graph shown in Figure 10 displays all the point-estimates for the different quantiles, with the random forest algorithm. One thing to notice is how close the predictions move in unison with the underlying data. This gives a good picture of how well the model works with the EURUSD data.

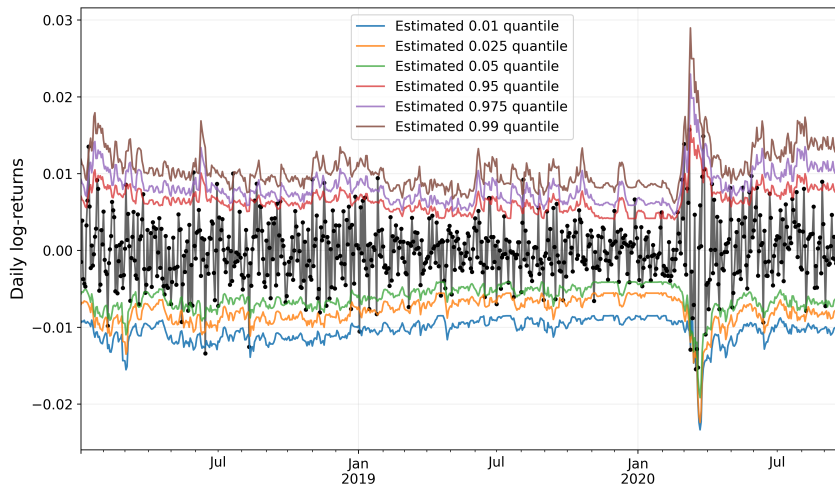


Figure 10: Random forest model. The black series is the daily log returns

Figure 11 shows a distribution of the Shap-values calculated from the random forest ensemble method. First, the plot indicates symmetry in the estimation of the quantiles. Second, risk reversal seems to be the significant model parameter as it shows both the substantial density and the greatest magnitude in the figure. In this model, high values for the risk reversal has a positive contribution to the prediction, and low values have a negative contribution. Excluding the outliers in the implicit volatility, the high/ low values have the same contribution has seen for the risk reversal.

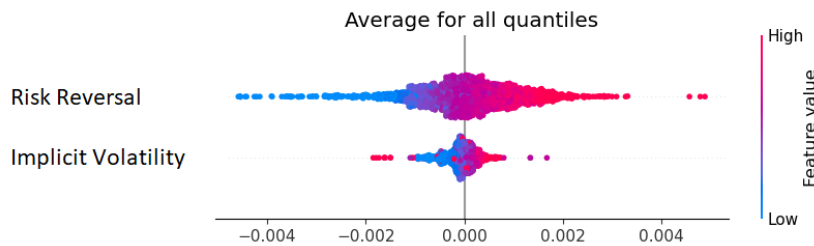


Figure 11: Random forest variable importance

## 5.2.2 Extreme Gradient Boosting

We have created a competing data set for training the model using gradient boosting. Both this data set and the data set created by the QR-IM model from Equation (4), are used for training.

### Training individual dataset

We use an ordinary gradient boosting algorithm to create a data set, exploring whether this model can outperform the QR-IM model.

In Table 4 we see the Breach Ratio. The performance is not as accurate as the QR-IM model.

quantile - $\alpha$	1.0 %	2.5 %	5.0 %	95.0 %	97.5 %	99.0 %
<i>Breach Ratio</i>	1.08	2.62	4.92	94.94	97.29	98.92

Table 4: Gradient Boost training set performance

### Validating and testing performance - XGBoost model 1

The first XGBoost prediction model is trained with the QR-IM data set, and the output is displayed in Figure 12 and Table 5. Again, the model performs better on the higher  $\alpha$ -quantiles. This is similar to the random forest algorithm in the previous section. Once more, the model shows excellent prediction power when modelling the spike behaviour of the data. This is evident from the small values for the *sum if breach* values in the 1% and 99% quantiles.

quantile - $\alpha$	1.0 %	2.5 %	5.0 %	95.0 %	97.5 %	99.0 %
<i>Breach Ratio</i>	0.7	2.1	5.47	95.09	97.9	99.44
<i>Sum if breach</i>	-0.006	-0.019	-0.052	0.064	0.023	0.005
<i>Sum if no breach</i>	-7.442	-5.680	-4.521	4.756	6.028	7.643
<i>Min. VaR</i>	-0.024	-0.022	-0.019	0.004	0.004	0.006
<i>Max. VaR</i>	-0.007	-0.006	-0.004	0.017	0.024	0.030

Table 5: XGBoost model 1 performance

The graph shown in Figure 12 displays all the point-estimates for the different quantiles, made by the XGBoost model 1. Again, the predictions move in unison with the underlying data, and clearly gives a good picture of on how well the model works with the EURUSD data.

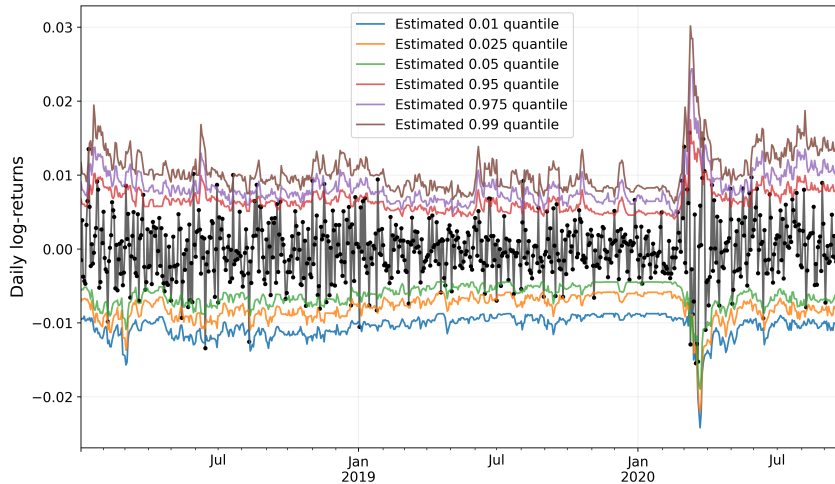


Figure 12: XGBoost model 1. The black series is the daily log returns

Figure 13 shows the Shap-values for the model. Again we see symmetry in the estimation of the quantiles. We also note that risk reversal is the most important explanatory parameter in the model. However, the explainable power is less eminent here, than with the random forest algorithm.

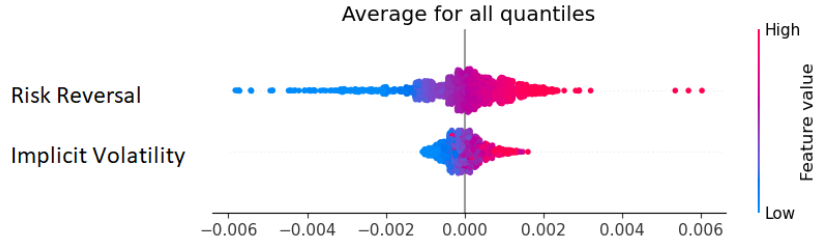


Figure 13: XGBoost model 1 variable importance.

### Validation and testing performance - XGBoost model 2

The second XGBoost prediction model we employ, is trained with the Gradient Boost generated data set, and the output is displayed in Table 5. We note that this model clearly is outperformed by the first extreme boosting model. The breach ratios are more remote from the desired threshold breach ratios - being the  $\alpha$ -quantile values. Also, *the sum if breach* is larger compared to the first extreme boosting model. Thus, it is safe to conclude that the QR-IM-generated data set outperforms the gradient boost-generated data set. Therefore the gradient boost-generated data set is not considered for further modelling.

quantile - $\alpha$	1.0 %	2.5 %	5.0 %	95.0 %	97.5 %	99.0 %
<i>Breach Ratio</i>	1.26	3.65	7.01	95.37	98.46	99.16
<i>Sum if breach</i>	-0.025	-0.056	-0.184	0.112	0.040	0.020
<i>Sum if no breach</i>	-5.860	-5.152	-3.612	4.155	5.477	6.391
<i>Min. VaR</i>	-0.015	-0.015	-0.011	0.001	0.002	0.003
<i>Max. VaR</i>	-0.005	-0.002	-0.001	0.022	0.025	0.019

Table 6: XGBoost model 2 performance

The Shap-values in Figure 14 indicate a more caotic behaviour compared to the Shap-values of the random forest and first extreme boosting model. These Shap-values show less of a dominant predictor.

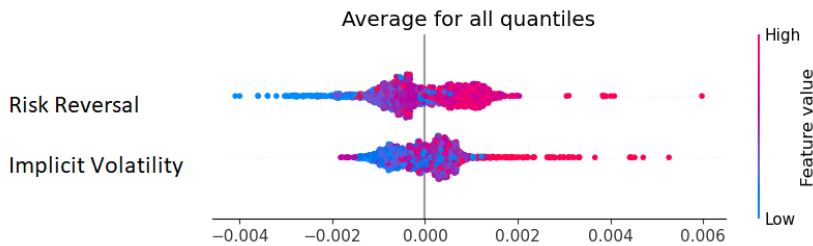


Figure 14: XGBoost model 2 variable importance

### 5.2.3 Light Gradient-Boosting Machine

In this section the light gradient-boosting machine has been used to create a competing data set for training the model. Both this data set and the data set created by the QR-IM model from Equation (4), are used for training.

### Traning individual data set

First we employ an ordinary gradient boosting algorithm to create the additional data set to see whether it can outperform the QR-IM model.

EURUSD	1.0 %	2.5 %	5.0 %	95.0 %	97.5 %	99.0 %
<i>Breach Ratio</i>	1.67	2.62	5.06	95.71	97.7	98.74

Table 7: LGBM created data set performance

### Validation and testing performance - LGBM model 1

Our first version of the LGBM prediction model is trained with the QR-IM dataset, and the output is displayed in Figure 15 and Table 8. This model also produces similar forecasting results to the random forest and extreme gradient boosting models. The best performance is accomplished at the 95%, 97.5% and 99% quantiles. The LGBM model yields good results for the *sum if breach*, indicating that the model is good at predicting spike behaviour in the out of sample data.

quantile - $\alpha$	1.0 %	2.5 %	5.0 %	95.0 %	97.5 %	99.0 %
<i>Breach Ratio</i>	0.55	1.82	5.33	95.09	97.62	99.44
<i>Sum if breach</i>	-0.004	-0.017	-0.053	0.066	0.022	0.004
<i>Sum if no breach</i>	-7.445	-5.668	-4.500	4.748	6.005	7.691
<i>Min. VaR</i>	-0.024	-0.022	-0.018	0.004	0.004	0.006
<i>Max. VaR</i>	-0.008	-0.006	-0.004	0.015	0.021	0.026

Table 8: LGBM model 1 performance

The graph shown in Figure 15 displays all the point-estimates for the different quantiles, made by the first LightGBM model. Once again, the predicted quantiles match the underlying data. We can see from the graph that the model performs well capturing the peak values.

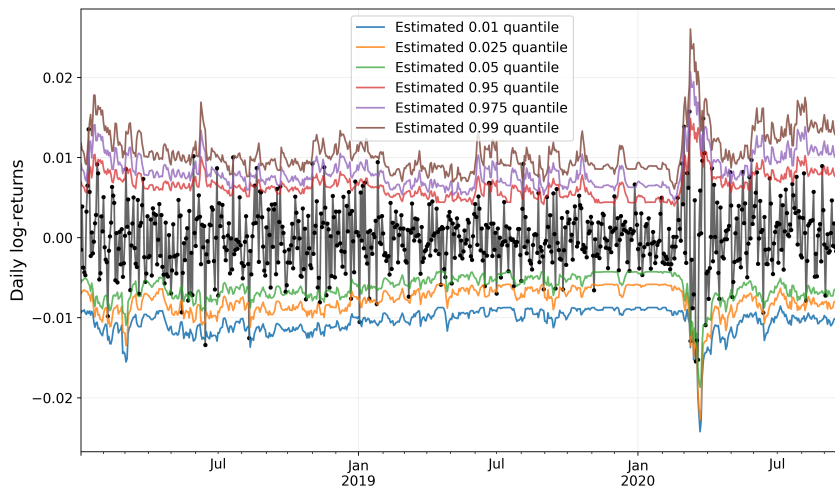


Figure 15: LGBM model 1. The black series is the daily log returns

Figure 16 shows a distribution of the Shap-values calculated from the LGBM ensemble method. The values indicates symmetry in the estimation of the quantiles. Furthermore, the risk reversal seems to be the significant model parameter as it shows both substantial density and the greatest

magnitude in the figure. The Shape values are very similar to the Shape values of the random forest model.

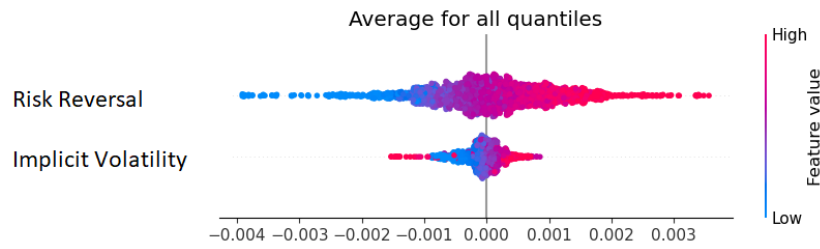


Figure 16: LGBM model 1 variable importance

### Hyper-parameter tuning

Employing the LGBM model, the user has the option of tuning wide range of hyper parameters, based on the desired outcome. In this particular problem the objective is to improve the 5%-quantile.

First, a new parameter is added to the input data set. This parameter called *movement*, is an interaction variable between the *At-the-money volatility* variable and the *25 delta RR* variable. This new variable is included to test the impact of new data on the estimates, and to generate information flow between the parameters. This is the same value used in hyper parameter tuning of the neural networks explained in Section 4.9.1.

The tuning is done employing a Python package called *verstack* [49], which runs different combinations of the parameters and chooses the best fit based on the test data. The training and test period is shown in Figure 17.

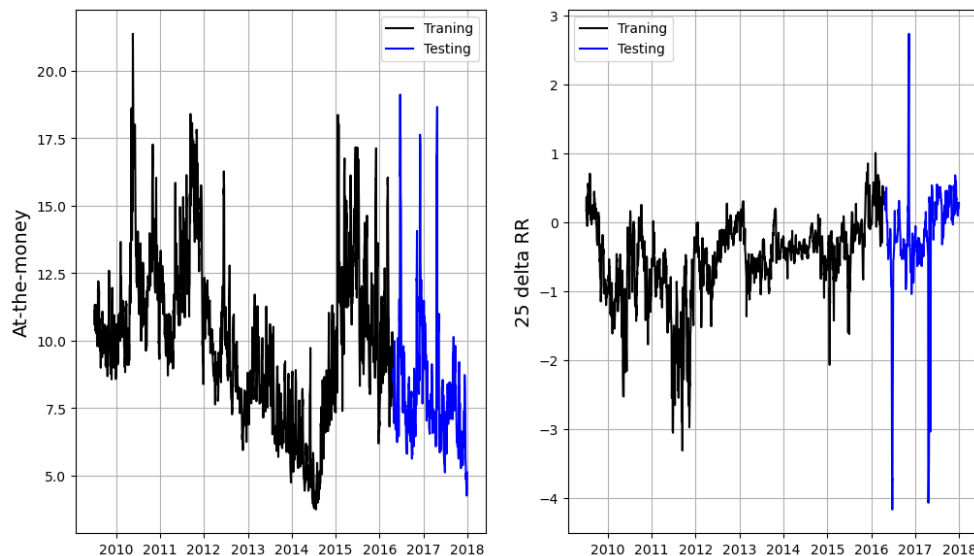


Figure 17: Training and testing period used for hyper-parameter tuning.

### Validation and testing performance - LGBM model 2

This leads to the second LGBM prediction model. This model is also trained with the QRIM dataset. The output is displayed in Figure 18 and Table 9. Compared to the first LGBM model, this second model, as expected, is better at predicting the 5% quantile. However, this comes at a price of a less accurate 97.5% quantile estimate. This model, so far, is the most accurate when predicting the three lower quantiles. Furthermore, this model, as XGBoost and random forest models, yields excellent results for modeling the spike behavior in the data.

quantile - $\alpha$	1.0 %	2.5 %	5.0 %	95.0 %	97.5 %	99.0 %
<i>Breach Ratio</i>	0.7	2.1	5.05	95.23	97.76	99.44
<i>Sum if breach</i>	-0.005	-0.019	-0.055	0.060	0.020	0.005
<i>Sum if no breach</i>	-7.445	-5.682	-4.526	4.816	6.046	7.712
<i>Min. VaR</i>	-0.024	-0.023	-0.019	0.00404	0.005	0.006
<i>Max. VaR</i>	-0.008	-0.005	-0.004	0.016	0.022	0.027

Table 9: LGBM model 2 performance

The graph shown in Figure 18 displays all the point-estimates for the different quantiles, made by the second LightGBM model. We see similar results as for the first model.

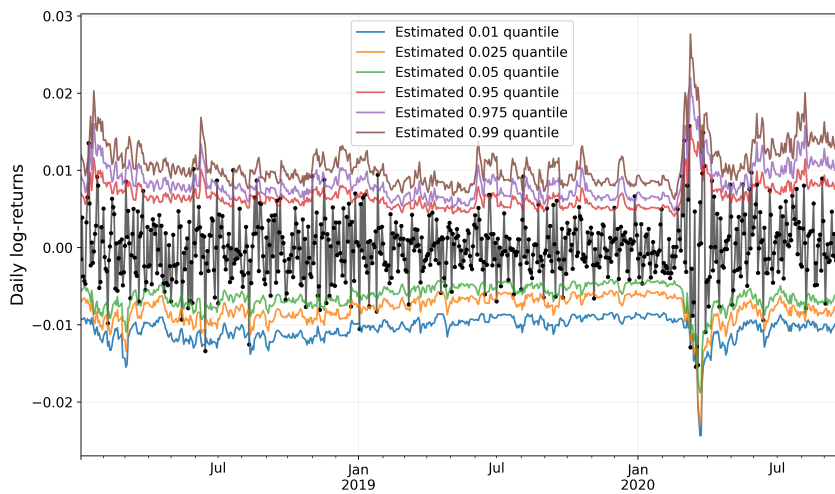


Figure 18: LGBM model 2. The black series is the daily log returns

Figure 19 shows the three variables used in the second LGBM model. Again, the symmetry is notable in the Shap plot. The biggest difference, compared to the second LGBM model, is the impact of the implicit volatility parameter. By introducing the link between the two parameters, the *movement* parameter has stolen some predictive power from the other parameters. As indicating by the figure, implicit volatility is far less important in this prediction.

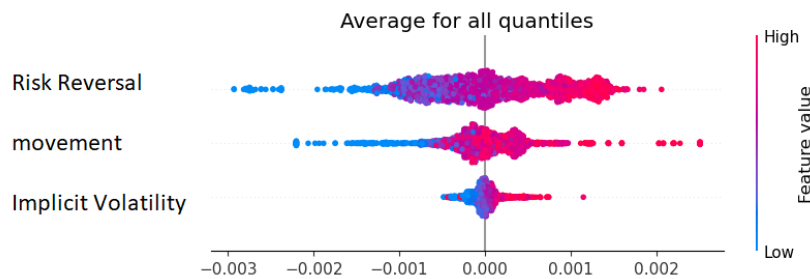


Figure 19: LGBM model 2 variable importance

### Validation and testing performance - LGBM model 3

The third LGBM prediction model is trained with the LGBM generated dataset, and the output is displayed in Table 10.

First, this model clearly is outperformed by the two other light boosting-machine models. The breach ratios are more remote from the desired threshold breach ratios - being the  $\alpha$ -quantile values. Second, *the sum if breach* is more substantial compared to the first extreme boosting model. Thus, it is safe to conclude that the QR-IM-generated data set outperforms the LGBM-generated data set, and is not considered for further modelling.

quantile - $\alpha$	1.0 %	2.5 %	5.0 %	95.0 %	97.5 %	99.0 %
<i>Breach Ratio</i>	1.4	4.49	8.42	95.65	96.63	99.3
<i>Sum if breach</i>	-0.020	-0.054	-0.104	0.067	0.033	0.010
<i>Sum if no breach</i>	-6.255	-4.860	-4.110	4.929	5.657	7.164
<i>Min. VaR</i>	-0.014	-0.013	-0.012	0.003	0.003	0.007
<i>Max. VaR</i>	-0.006	-0.004	-0.003	0.018	0.019	0.021

Table 10: LGBM model 3 performance

The Shap-values in Figure 14 indicate a more chaotic behaviour compared to the Shap-values seen for the two other LGBM models. For the first time both variables have similar predictive power in the model. The Shap values are clearly skewed and not symmetric.

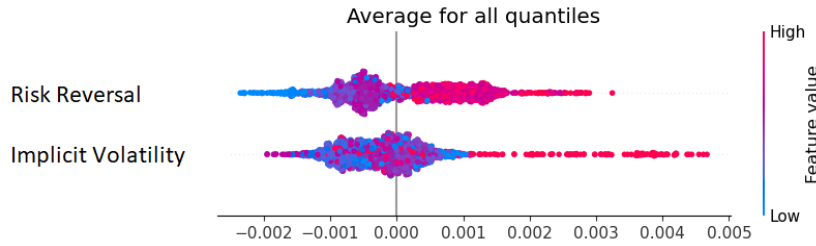


Figure 20: LGBM model 3 variable importance

#### 5.2.4 Categorical Boosting

The categorical boosting prediction model is trained with the QR-IM data set, and the output is displayed in Figure 21 and Table 11. Once again, the model yields similar results to those generated by the best models presented above. However, this model is the best at predicting the 2.5% quantile.

Implicit Volatility

quantile - $\alpha$	1.0 %	2.5 %	5.0 %	95.0 %	97.5 %	99.0 %
<i>Breach Ratio</i>	0.7	2.23	5.47	94.95	97.62	99.44
<i>Sum if breach</i>	-0.006	-0.020	-0.056	0.064	0.022	0.005
<i>Sum if no breach</i>	-7.411	-5.641	-4.496	4.789	6.017	7.688
<i>Min. VaR</i>	-0.023	-0.022	-0.018	0.004	0.004	0.006
<i>Max. VaR</i>	-0.008	-0.005	-0.004	0.016	0.023	0.029

Table 11: CatBoost performance



The graph shown in Figure 21 displays all the point-estimates for the different quantiles, made by CatBoost model. The algorithm models the quantiles of the underlying data with good precision.

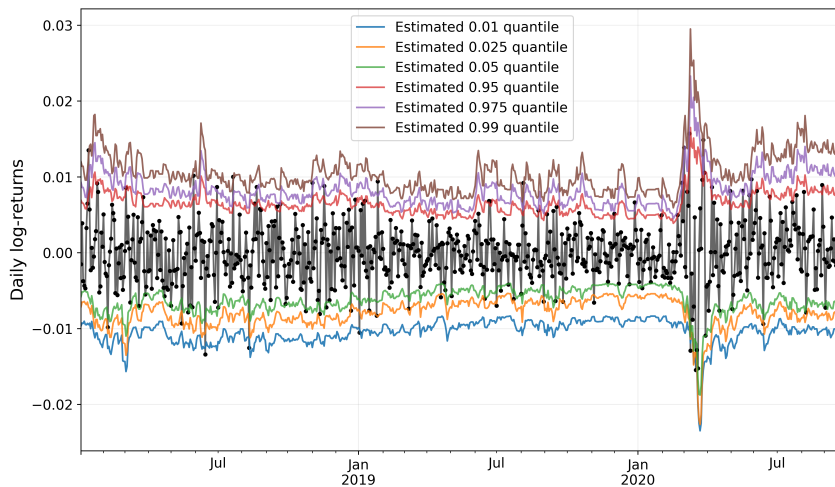


Figure 21: CatBoost model. The black series is the daily log returns

Figure 22 shows a distribution of the Shap-values calculated from the CatBoost ensemble method. First, the Shap values are very similar to those of the random forest model. Symmetry is once again present, as well as the risk reversal being the significant model parameter as it shows both the highest density and the greatest magnitude in the figure.

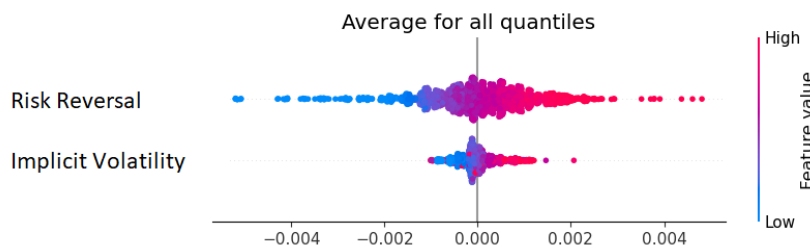


Figure 22: CatBoost model variable importance

### 5.2.5 Stacking

Stacking is similar to hyper parameter tuning, it is a tool used for improving model performance. As an example, a model based on a combination of the LGBM model 2 and the Catboost model is used to generate an improved model. This stacking model is a combination with 37% of the LGBM model 2 and 63% of the the CatBoost model. The model yields:

EURUSD	1.0 %	2.5 %	5.0 %	95.0 %	97.5 %	99.0 %
Breach Ratio	0.7	2.18	5.31	95.05	97.67	99.44

Table 12: 37% LGBM model 2 with 63% CatBoost, model performance

The stacking model leads to interesting results being a combining of two models. The model generates good over all predictions for all quantiles. Despite the results of this model, a further development of the stacking approach is not going to be looked into in further detail.

### 5.3 Neural network

We now examine neural networks. We explore different uses of the algorithms for feed forward neural network, recurrent neural network and the long short-term memory neural network. The algorithms are going to be tackled in the same order, starting with feed forward neural network.

For training and validation the data has been split in three parts. The first part starts at the beginning of the data set and lasts until the end of 2017. This part is used for training and testing the model. From the first part 80% of the data is used for training and the remaining for testing. While the second part starts in 2018 and lasts until the end of the data set. This part is used for the model validation. All three periods are illustrated in Figure 23.

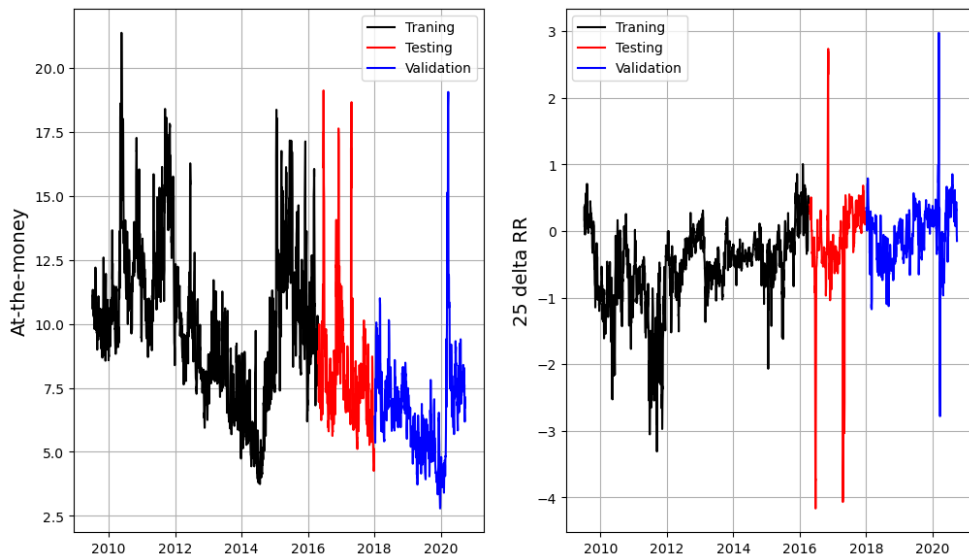


Figure 23: Neural network training period and validation period

All neural networks have problems modelling historical moments with high volatility. This can easily be seen in the figures that show the predictions and compare them to Figure 23 which shows the validation period. In periods of great volatility, the model is not able to capture this response. The three neural network models have all used the additional *movement* variable explained in Chapter 4.9.1 and also used in the hyper parameter tuning of the second LGBM model.

#### 5.3.1 Feed forward neural network

The feed forward neural network prediction model is trained with the QR-IM data set, and the output is displayed in Figure 24 and Table 13. First, we see that the model thinks the best prediction is straight lines. This is also present in the *sum if breach* and *sum if no breach* values, all of which are very high. This is an indication that the model is conservative in its predictions. The predicted quantiles from the ensemble methods were much closer to the modeled data, than those of this neural network.

quantile - $\alpha$	1.0 %	2.5 %	5.0 %	95.0 %	97.5 %	99.0 %
<i>Breach Ratio</i>	2.38	4.35	8.27	90.74	96.21	98.74
<i>Sum if breach</i>	-0.052	-0.098	-0.188	0.186	0.086	0.038
<i>Sum if no breach</i>	-9.477	-8.139	-6.752	6.420	7.917	9.873
<i>Min. VaR</i>	-0.013	-0.011	-0.009	0.008	0.011	0.0137
<i>Max. VaR</i>	-0.013	-0.011	-0.009	0.008	0.011	0.0137

Table 13: FFNN performance

The graph shown in Figure 24 displays all the point-estimates for the different quantiles, made by FFNN model. From the graph we can see the model predicts straight lines.

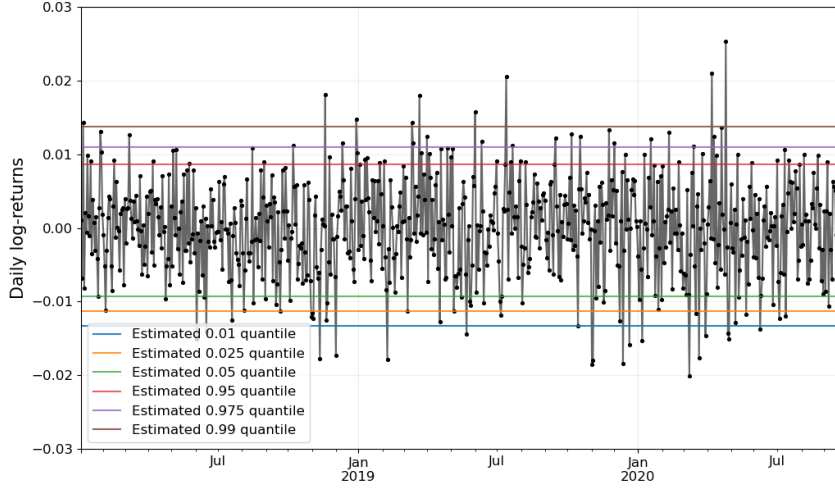


Figure 24: FFNN model. The black series is the daily log returns

### 5.3.2 Recurrent neural network

The recurrent neural network prediction model is trained with the QR-IM data set, and the output is displayed in Figure 25 and Table 14. Again, the model chooses straight lines as its best prediction. However, this time we notice an effort by the model to capture something different at the spikes in volatility, but it fails. Similar to the feed forward neural network this model is conservative in its predictions and actual data frequently exceeds the predicted quantiles, thus yielding high values for both the *sum if breach* and *sum if no breach* values.

quantile - $\alpha$	1.0 %	2.5 %	5.0 %	95.0 %	97.5 %	99.0 %
<i>Breach Ratio</i>	2.42	4.27	7.68	90.75	96.02	98.72
<i>Sum if breach</i>	-0.069	-0.157	-0.208	0.201	0.088	0.047
<i>Sum if no breach</i>	-9.139	-7.613	-6.547	6.278	7.849	9.570
<i>Min. VaR</i>	-0.018	-0.264	-0.009	0.008	0.011	0.013
<i>Max. VaR</i>	-0.013	-0.011	0.636	0.237	0.353	0.200

Table 14: RNN performance

The graph shown in Figure 25 displays all the point-estimates for the different quantiles, made by RNN model. From the graph we can see the model predicts straight lines. Furthermore, we see an explosion in quantile predictions around the high volatility occurrence in the beginning of March 2020 and the Covid19 outbreak.

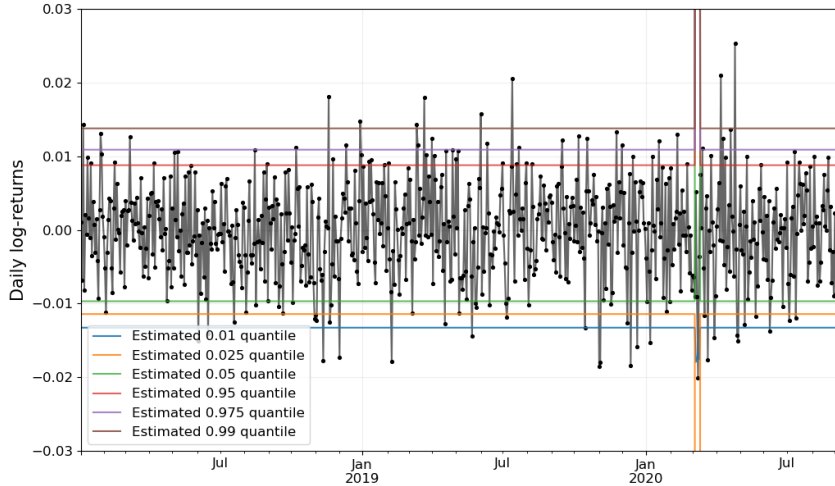


Figure 25: RNN model. The black series is the daily log returns

### 5.3.3 Long short-term memory

The long short-term memory neural network prediction model is trained with the QR-IM data set, and the output is displayed in Figure 26 and Table 15. The model shows small fluctuations, but close to straight lines. This model yields the smallest numbers for the *sum if breach* and the *sum if no breach* of all the neural networks. However, the model is still far from the prediction level of the ensemble methods.

quantile - $\alpha$	1.0 %	2.5 %	5.0 %	95.0 %	97.5 %	99.0 %
<i>Breach Ratio</i>	3.13	4.97	8.68	96.16	96.16	98.44
<i>Sum if breach</i>	-0.052	-0.095	-0.200	0.213	0.110	0.046
<i>Sum if no breach</i>	-9.303	-8.114	-6.529	6.132	7.346	9.242
<i>Min. VaR</i>	-0.028	-0.018	-0.012	0.007	0.010	0.013
<i>Max. VaR</i>	-0.012	-0.010	-0.006	0.011	0.013	0.035

Table 15: LSTM performance

The graph shown in Figure 25 displays all the point-estimates for the different quantiles, made by LSTM model. From the graph we can see the model predicts straight lines with some fluctuations around the mean of the line. Furthermore, we see an explosion in the predictions around the high volatility occurrence in the beginning of March 2020 and the Covid19 outbreak. However, this models is better at predicting volatility than the RNN model.

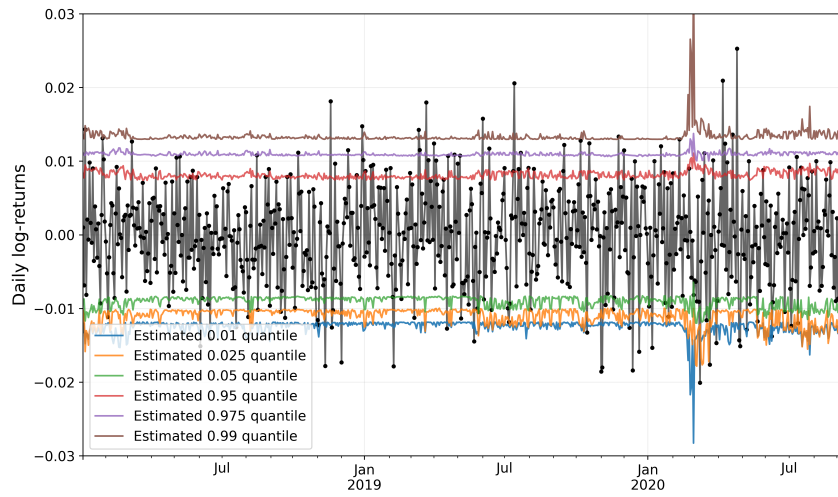


Figure 26: LSTM model. The black series is the daily log returns

## 5.4 Summary

In order to validate the performance of the different methods a base line model is needed. The chosen base line model is the QR-IM model, from the paper by de Lange, Ristad and Westgaard [2], when doing the out of sample estimates.

In general, the performance of the ensemble methods are more stable and consistent compared to the neural network. The ensemble methods handle the periods with outliers better than the the neural networks, whose predictions tend to explode around these events.

The ensemble models perform well, all in all, compared to the baseline QR-IM model. Three models stand out in comparison to the baseline model: LightGBM model 2, Categorical boost, and the stacking model between the two. These models all perform better or equal to that of the QR-IM model on the lower quantiles; 1%, 2.5% and 5%. At the higher qauntiles; 95%, 97.5% and 99%, the QR-IM model is better or equal to the three models.

The neural network models are in general quite unstable and is probably not ideally suited for this task. The models produce worse or equal quantile predictions compared to the QR-IM model for all quantiles.

All the models are summarized with their breach ratios for the out of sample data in Table 16.

quantile - $\alpha$	1.0 %	2.5 %	5.0 %	95.0 %	97.5 %	99.0 %
<b>Base line model</b>						
QR-IM	0.7	2.1	6.3	95	97.5	99.3
<b>Ensemble methods</b>						
Random Forest	0.7	2.1	5.75	95.09	97.62	99.44
XGB (1)	0.7	2.1	5.47	95.09	97.9	99.44
XGB (2)	1.26	3.65	7.01	95.37	98.46	99.16
LightGBM (1)	0.55	1.82	5.33	95.09	97.62	99.44
LightGBM (2)	0.7	2.1	5.05	95.23	97.76	99.44
LightGBM (3)	1.4	4.49	8.42	95.65	96.63	99.3
CatBoost	0.7	2.23	5.47	94.95	97.62	99.44
<b>Stacking Models</b>						
LGBM2&CatBoost	0.7	2.18	5.31	95.05	97.67	99.44
<b>Neural network</b>						
FFNN	2.38	4.35	8.27	90.74	96.21	98.74
RNN	2.42	4.27	7.68	90.75	96.02	98.72
LTSM	3.13	4.97	8.68	96.16	96.16	98.44

Table 16: Summary of all model breach ratios

We note that for *the sum if breach* parameter, less is better. By having a small sum, the model is better at predicting the outcome when a breach happens. However, if a small *sum if breach* value is combined with a high value for *sum if no breach* then the model may be too conservative in its predictions. In Table 17 both the *if breach* (i.b.) and *if no breach* (i.n.b.) sums are listed for all models. It is difficult to distinguish between the models and find a superior model. The random forest, XGB model 1 and CatBoost algorithms are all very similar when comparing the i.b. scores, all having an approximate score of 0.17. However, the random forest is significantly better when comparing the i.n.b scores, having the lowest score of 35.91. Thus, the random forest may be considered superior amongst the three. It is more difficult to distinguish between the LGBM model 1 and the LGBM model 2, each having its one superior trait. Thus the choice of model is left to the user, based on which feature *sum if breach* and *sum if no breach* one deems more important. The LGBM model 1 has the lowest i.b. score among all models, of 0.164.

quantile - $\alpha$		1.0 %	2.5 %	5.0 %	95.0 %	97.5 %	99.0 %	Sum
<b>Ensemble methods</b>								
Random Forest	i.b.	-0.005	-0.018	-0.054	0.067	0.022	0.005	0.171
	i.n.b.	-7.421	-5.641	-4.483	4.729	5.982	7.650	35.91
XGB (1)	i.b.h	-0.006	-0.019	-0.052	0.064	0.023	0.005	0.169
	i.n.b.	-7.442	-5.680	-4.521	4.756	6.028	7.643	36.07
XGB (2)	i.b.	-0.025	-0.056	-0.184	0.112	0.040	0.020	0.437
	i.n.b.	-5.860	-5.152	-3.612	4.155	5.477	6.391	30.65
LightGBM (1)	i.b.	-0.004	-0.017	-0.053	0.066	0.022	0.004	0.166
	i.n.b.	-7.445	-5.668	-4.500	4.748	6.005	7.691	36.06
LightGBM (2)	i.b.	-0.005	-0.019	-0.055	0.060	0.020	0.005	0.164
	i.n.b.	-7.445	-5.682	-4.526	4.816	6.046	7.712	36.23
LightGBM (3)	i.b.	-0.020	-0.054	-0.104	0.067	0.033	0.010	0.288
	i.n.b.	-6.255	-4.860	-4.110	4.929	5.657	7.164	32.97
CatBoost	i.b.	-0.006	-0.020	-0.056	0.064	0.022	0.005	0.173
	i.n.b.	-7.411	-5.641	-4.496	4.789	6.017	7.688	36.04
<b>Neural network</b>								
FFNN	i.b.	-0.052	-0.098	-0.188	0.186	0.086	0.038	0.648
	i.n.b.	-9.477	-8.139	-6.752	6.420	7.917	9.873	48.57
RNN	i.b.	-0.069	-0.157	-0.208	0.201	0.088	0.047	0.770
	i.n.b.	-9.120	-8.486	-6.287	6.274	8.406	10.064	47.00
LTSM	i.b.	-0.052	-0.095	-0.200	0.213	0.110	0.046	0.716
	i.n.b.	-9.303	-8.114	-6.529	6.132	7.346	9.242	46.66

Table 17: Summary of all model if breach values, i.b. and the if not breach values, i.n.b

In Table 18, all the models are summarized with their Christoffersen test p-values [50] and DQ-test p-values with four lags [51]. Both the XGB model 2 and LightGBM model 3 rejects the null hypotheses. Thus, the observed number of VaR breeches is significantly different from the expected number of breeches for the two models. The remaining models generally display high p-values, indicating high forecasting performance across the quantiles.

<b>Christoffersen test (p-value)</b>	1.0 %	2.5 %	5.0 %	95.0 %	97.5 %	99.0 %
Random Forest	0.398	0.489	0.364	0.364	0.846	0.200
XGB (1)	0.398	0.489	0.564	0.917	0.489	0.200
XGB (2)	0.175	0.000	0.000	0.000	0.157	0.001
LightGBM (1)	0.200	0.343	0.810	0.917	0.846	0.200
LightGBM (2)	0.398	0.489	0.945	0.781	0.660	0.200
LightGBM (3)	0.306	0.002	0.000	0.419	0.157	0.398
CatBoost	0.398	0.660	0.564	0.945	0.846	0.200
<b>DQ test (p-value)</b>	1.0 %	2.5 %	5.0 %	95.0 %	97.5 %	99.0 %
Random Forest	0.622	0.929	0.191	0.727	0.255	0.965
XGB (1)	0.806	0.944	0.192	0.280	0.858	0.965
XGB (2)	0.128	0.000	0.000	0.000	0.084	0.001
LightGBM (1)	0.673	0.938	0.319	0.648	0.717	0.964
LightGBM (2)	0.617	0.942	0.491	0.381	0.201	0.963
LightGBM (3)	0.000	0.001	0.000	0.951	0.506	0.991
CatBoost	0.692	0.943	0.371	0.678	0.204	0.964

Table 18: Summary of all model containing p-values for the Christoffersen test and DQ-test (with four lags).

---

## 6 Conclusion

In this study we employ ensemble methods and neural networks to forecast Value at Risk for daily exchange rates. The ensemble methods, which are forward-looking and utilizes directly observable option prices as explanatory variables, show great potential at predicting the daily exchange rate. The neural network, though also forward-looking and utilizing directly observable option prices as explanatory variables, does not perform at a desired prediction level.

Six key outtakes from the models

- The ensemble methods are better at predicting the spike behavior of the EURUSD than the neural networks. The ensemble methods are well suited for predicting the daily EURUSD currency cross distribution, including its VaR.
- Neural networks could benefit from more training data and perhaps a better model architecture. It is clear that the neural network can improve a lot compared to the ensemble methods. One way to do this is by introducing more layers and more nodes. However, by doing so, the model becomes more of a black box.
- The advantage of neural networks compared to ensemble methods is the way they estimate the quantiles. The neural network is constructed such that all quantiles can be estimated simultaneously. On the other hand, the ensemble methods forecast one quantile at a time. For huge data, the possibility of computing all quantiles simultaneously can be a significant factor when it comes to computational time.
- Model stacking and hyperparameter tuning significantly improved the models in terms of the over all performance of the breach ratio.
- The second LightGBM model, Categorical boost, and the stacking model between the two, stand out when comparing the breach values to the base line, QR-IM, model. These models all perform better or equal at the lower quantiles and have less or equal performance at the higher qauntiles.
- The random forest and LightGBM model 1 and 2 have the best performances among the stand alone models, in terms of i.b. and i.n.b. values.

### Future research

Several possible directions can be taken to further examine the models explored in the work. First, a potential improvement would be to test the models' performance in less efficient markets than EURUSD. Second, including more explanatory variables, such as macroeconomic variables, might improve the models. This is very useful for machine learning methods, as they benefit from more training data. Thirdly, one could try different types of combinations of layers in the neural network. This could lead to better forecasts and more stable results. Lastly, ensemble methods might be improved from exploring more hyperparameter tuning and stacking methods.



---

## References

- [1] J. Schaumburg, ‘Predicting extreme value at risk: Nonparametric quantile regression with refinements from extreme value theory’, *Computational Statistics and Data Analysis*, 2012.
- [2] P. De Lange and M. Risstad, ‘Estimating value-at-risk using quantile regression and implied moments’, Sep. 2022.
- [3] R. Engle and S. Manganelli, ‘Caviar: Conditional autoregressive value at risk by regression quantiles’, 1999.
- [4] J. W. Taylor, ‘Using exponentially weighted quantile regression to estimate value at risk and expected shortfall’, *Journal of financial Econometrics*, 2008.
- [5] M. Chen and J. Chen, ‘Application of quantile regression to estimation of value at risk.’, *Review of Financial Risk Management*, 2002.
- [6] J. W. Taylor, ‘A quantile regression approach to estimating the distribution of multiperiod returns.’, *Journal of Derivatives*, 1999.
- [7] A. Y. Huang, S. P. Peng, F. Li and C. J. Ke, ‘Volatility forecasting of exchange rate by quantile regression.’, *International Review of Economics & Finance*, 1999.
- [8] J. Jeon and J. W. Taylor, ‘Using caviar models with implied volatility for value-at-risk estimation.’, *Journal of Forecasting*, 2013.
- [9] B. Y. Chang, P. Christoffersen and K. Jacobs, ‘Market skewness risk and the cross section of stock returns.’, *Journal of Financial Economics*, 2013.
- [10] G. Barone-Adesi, C. Legnazzi and C. Sala, ‘Option-implied risk measures: An empirical examination on the s&p 500 index.’, *International Journal of Finance & Economics*, 2019.
- [11] M. Huggenberger, C. Zhang and T. Zhou, ‘Forward-looking tail risk measures’, 2018.
- [12] F. Bossens, G. Rayée, N. S. Skantzos and G. Deelstra, ‘Vanna-volga methods applied to fx derivatives: From theory to market practice.’, *International Journal of Theoretical and Applied Finance*, 2010.
- [13] M. Sarma, S. Thomas and A. Shah, ‘Selection of value-at-risk models.’, *Journal of Forecasting*, 2003.
- [14] A. Bijelic and T. Ouijjane, *Predicting exchange rate value-at-risk and expected shortfall: A neural network approach*, 2019.
- [15] Z. Xu, Y. Zeng, Y. Xue and S. Yang, ‘Foreign exchange prediction using machine learning approach: A pilot study’, *Computational Economics*, 2021.
- [16] Y. Heryadi and A. Wibowo, ‘Foreign exchange prediction using machine learning approach: A pilot study’, *International Conference on Information and Communications Technology*, 2021.
- [17] Z. Xu, Y. Zeng, Y. Xue and S. Yang, ‘Forecasting exchange rate value at risk using deep belief network ensemble based approach’, *Procedia Computer Science*, 2018.
- [18] G. Petneházi, ‘Quantile convolutional neural networks for value at risk forecasting.’, 2021.
- [19] D. Pradeepkumar and V. Ravi, ‘Forecasting financial time series volatility using particle swarm optimization trained quantile regression neural network.’, *Applied Soft Computing*, 2017.
- [20] J. W. Taylor, ‘A quantile regression neural network approach to estimating the conditional density of multiperiod returns’, *Journal of Forecasting*, 2000.
- [21] Q. Xu, X. Liu, C. Jiang and K. Yu, ‘Quantile autoregression neural network model with applications to evaluating value at risk.’, *Applied Soft Computing*, 2016.
- [22] J. Yen, X. Chen and K. K. Lai, ‘A statistical neural network approach for value-at-risk analysis.’, *International Joint Conference on Computational Sciences and Optimization*, 2009.
- [23] X. Yan, W. Zhang, L. Ma, W. Liu and Q. Wu, ‘Parsimonious quantile regression of financial asset tail dynamics via sequential learning.’, *In Advances in neural information processing systems*, 2015.

- 
- [24] K. Kakade, I. Jain and A. K. Mishra, ‘Value-at-risk forecasting: A hybrid ensemble learning garch-lstm based approach’, *Resources Policy*, 2022, ISSN: 0301-4207. DOI: <https://doi.org/10.1016/j.resourpol.2022.102903>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0301420722003476>.
- [25] M. Andreani, V. Candila and L. Petrella, *Quantile Regression Forest for Value-at-Risk Forecasting Via Mixed-Frequency Data*, M. Corazza, C. Perna, C. Pizzi and M. Sibillo, Eds. Cham: Springer International Publishing, 2022, ISBN: 978-3-030-99638-3.
- [26] F. Jiang, W. Wu and Z. Peng, ‘A semi-parametric quantile regression random forest approach for evaluating multi-period value at risk’, 2017.
- [27] K. Gorgen, J. Meirer and M. Schienle, *Predicting value at risk for cryptocurrencies with generalized random forests*, 2022. DOI: 10.48550/ARXIV.2203.08224. [Online]. Available: <https://arxiv.org/abs/2203.08224>.
- [28] X. Cai, Y. Yang and G. Jiang, ‘Online risk measure estimation via natural gradient boosting’, 2020. DOI: 10.1109/WSC48552.2020.9383934.
- [29] F. Rosenblatt, ‘The perceptron - a perceiving and recognizing automaton’, Jan. 1957. [Online]. Available: <https://www.nature.com/articles/323533a0>.
- [30] D. E. Rumelhart, G. E. Hinton and R. J. Williams, ‘Learning representations by back-propagating errors.’, 1986. [Online]. Available: <https://doi.org/10.1038/323533a0>.
- [31] S. Sun, Z. Cao, H. Zhu and J. Zhao, ‘A survey of optimization methods from a machine learning perspective’, 2019. [Online]. Available: <http://arxiv.org/abs/1906.06821>.
- [32] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.
- [33] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. DOI: 10.48550/ARXIV.1412.6980. [Online]. Available: <https://arxiv.org/abs/1412.6980>.
- [34] Y. LeCun, L. Bottou and K. R. Orr G. B. and Muller, *Efficient BackProp - Neural Networks: tricks of the trade*. Springer, 1998.
- [35] D. Fabrice, *Financial Time Series Data Processing for Machine Learning*. Paris: Artificial Intelligence Department of Lysis, 2019.
- [36] Scikit-learn, *Random forest*. [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html#sklearn.ensemble.RandomForestRegressor>.
- [37] Scikit-learn, *Gradient boost*. [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html#sklearn.ensemble.GradientBoostingRegressor>.
- [38] Scikit-learn, *Adaboost*. [Online]. Available: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html#sklearn.ensemble.AdaBoostRegressor>.
- [39] Microsoft, *Light gradient-boosting machine*. [Online]. Available: <https://lightgbm.readthedocs.io/en/v3.3.2/>.
- [40] DMLC, *Extreme gradient boosting*. [Online]. Available: <https://xgboost.readthedocs.io/en/stable/>.
- [41] Yandex, *Categorical boosting*. [Online]. Available: <https://catboost.ai/>.
- [42] Keras. [Online]. Available: <https://keras.io/>.
- [43] G. Zhang, B. Patuwo and M. Hu, ‘Forecasting with artificial neural networks: The state of the art’, *International Journal of Forecasting*, 1998.
- [44] J. H. Friedman, T. Hastie and R. Tibshirani, *The Elements of Statistical Learning*. Springer, 2001.
- [45] S. Gu, B. Kelly and D. Xiu, ‘Empirical asset pricing via machine learning’, 2020.
- [46] M. Boyd and I. Kastr, ‘Designing a neural network for forecasting financial and economic time series’, 1995.
- [47] Keras, *Callbacks api*. [Online]. Available: <https://keras.io/api/callbacks/>.
- [48] L. S. Shapley, ‘Notes on the n-person game – ii: The value of an n-person game’, 1951.
- [49] D. Zherebtsov, *Verstack*. [Online]. Available: <https://pypi.org/project/verstack/>.

- 
- [50] P. F. Christoffersen, 'Evaluating interval forecasts', *International Economic Review*, 1998.
- [51] R. F. Engle and S. Manganelli, 'Caviar: Conditional autoregressive value at risk by regression quantiles', *Journal of Business & Economic Statistics*, vol. 22, 2004.



 **NTNU**

Norwegian University of  
Science and Technology