A Neural Network with Shared Dynamics for Multi-Step Prediction of Value-at-Risk and Volatility

Nalan Baştürk
Maastricht University
n.basturk@maastrichtuniversity.nl

Peter C. Schotman
Maastricht University
p.schotman@maastrichtuniversity.nl

Hugo Schyns*
Maastricht University
h.schyns@maastrichtuniversity.nl

April 19, 2022

Abstract: We develop a Long Short-Term Memory (LSTM) neural network for the joint prediction of volatility, realized volatility and Value-at-Risk. Regularization by means of pooling the dynamic structure for the different outputs of the models is shown to be a powerful method for improving forecasts and smoothing Value-at-Risk (VaR) estimates. The method is applied to daily and high-frequency returns of the S&P500 index over a period of 25 years.

Keywords: Neural Network, Value-at-Risk, Volatility Models, Equity Returns, Risk Management

JEL codes: G17 (Financial Forecasting and Simulation), C32 (Time-Series Models; Dynamic Quantile Regressions; Dynamic Treatment Effect Models; Diffusion Processes; State Space Models)

*Corresponding author, Tongersestraat 53, 6211 LM Maastricht, h.schyns@maastrichtuniversity.nl

1 Introduction

Value-at-Risk (*VaR*) remains one of the key risk management tools. As a risk measure it combines elements of conditional volatility with the shape of the left tail of a return distribution. Estimates of *VaR* are highly nonlinear functions of past return data and other predictive variables. Since volatility is a crucial scaling variable for constructing *VaR*, the performance of a *VaR* model is closely related to the quality of a volatility prediction model. For volatility models nonlinearities have been investigated since the early GARCH models with leverage effects.

In recent years much progress has been made in modeling the nonlinearities using neural network (NN) models and other machine learning methods. Examples for *VaR* are the early contribution of Taylor (1999), or more recently Wu and Yan (2019). Similarly, Christensen, Siggaard, and Veliyev (2021) and Rahimikia and Poon (2020) report that NN models outperform Corsi's (2009) HAR for predicting realized variance, while Bucci (2020) finds that recurrent NN models pick up important nonlinearities for improved forecasting of monthly realized volatility.

Our model combines the two strands of literature. We develop a Neural Network (NN) model for multiple outputs that jointly predicts realized volatility, the conditional variance of daily close-to-close returns as well as *VaR* at different quantiles and for multiple horizons. The main motivation for the multiple output model is that we can pool the dynamic specification for the different variables we predict. Realized variance is obviously related to the conditional daily variance. Both realized variance and conditional daily variance measure the crucial scaling factor for *VaR* estimates given the quantiles of scaled returns. We thus expect that the latent variables ('neurons') that predict realized variance and conditional daily variance are useful for *VaR* and the other variables.

Our approach differs from a number of other applications of NN to volatility modelling in the literature. Donfack and Dufays (2020) introduce NN's to allow for time-varying parameters in a GARCH model as an extension of the Engle and Rangel (2008) approach for modeling the unconditional volatility as a function of macro variables. Bucci (2020) considers a variety of NN architectures to predict monthly realized volatility using macroeconomic predictors. The main innovation of our approach is the multi-output design by simultaneously modeling volatility and *VaR*, gaining predictive power from the pooling of dynamic structure in the LSTM. A second difference with the setup in Bucci (2020) is the length of the input sequences in the LSTM. We view our approach as most closely related to Corsi (2009). Corsi's HAR model forecasts future volatility as a parsimonious linear combination of three predictors. Corsi (2009) specifies the predictors as intuitively motivated weighted averages of past realised volatility. Our NN model replaces these by data-driven nonlinear functions of past realised volatilities and leverage effects.

The multi-output design is a distinguishing feature of our approach. Combining forecasts for related variables puts regularization on the NN architecture, since the dynamic part of the model is forced to have predictive power for the different outputs. It is only for the final step from the last layer of hidden states to the different outputs that we allow for an output specific link. This means that the bulk of the parameters in the network is common to all the different outputs.

Neural network models offer great flexibility, but are often criticized for their black box type predictions. The model we propose retains an interpretable structure. We specify a recurrent NN, more precisely a Long Short Term Memory (LSTM) model with a single hidden layer and a linear output. The LSTM model has a very flexible dynamic specification. It processes long histories of past data, in our case realized volatilities and returns, to construct a number of latent variables that serve as predictors for the final outputs. The neurons themselves are highly nonlinear functions of long past sequences of returns and realized variances. With the LSTM we have both great flexibility, but also parsimony. In the application to equity returns we find that a small set of neurons is often enough to linearly predict both volatility as well as *VaR*.

The structure of the proposed LSTM is motivated by the HAR and AHAR models introduced by Corsi (2009) and Corsi et al. (2012). In these models it appears that a few well designed temporal aggregations, such as weekly and monthly realized variance and the negative part of weekly and monthly returns, have strong predictive power. Bollerslev et al. (2016) follow this heterogenous structure and improve the HAR model by incorporating measurement error into their HARQ model. In LSTM such temporally aggregated constructs are learned by the network, simple averages being a very special case. The non-parametric weighting scheme that the LSTM applies to past observations and their nonlinear transformations show that our model can be thought of as a generalized version of the HAR model.

LSTM models are part of the family of Recurrent Neural Networks (RNN). Recurrent NN's create memory in the model. The special feature of the LSTM is the sharing of parameters when processing each time step in a long sequence of temporally ordered past data. The structure of a LSTM, introduced by Hochreiter and Schmidhuber (1997), avoids problems with vanishing and exploding gradients for parameters associated with data in the distant past. In the textbook specification we adopt, the bulk of the parameters in the network is associated with the dynamic structure, while each of the outputs is related to the hidden layer neurons with a few parameters. Moreover, the number of parameters does not depend on the length of the histories of past data used to predict next day's events.

With the multi-output model we are also able to incorporate recent developments in volatility modeling. We extend the HEAVY model of Shephard and Sheppard (2010) who advocate a joint model for squared daily close-to-close returns and the realized variance of open-to-close returns. The realized variance is an accurate measurement of volatility, but misses the overnight returns, whereas the daily squared return is a very noise measure of volatility, but cover the full day. Therefore squared daily returns and realized variance are two of the outputs in our neural network model.

An important element in the design of the model is the choice of a loss function. Most of the NN literature uses Mean-Squared-Error (MSE) loss. For *VaR* this does not work. Since realized *VaR* is not observed, we will need to estimate the *VaR* variables using a quantile regression function. Linear quantile regression is well known, see Koenker and Bassett (1978). White (1992) shows that non-linear quantile regressions using a neural network provide consistent estimates. In an early application, Taylor (1999) implements such a neural network model for *VaR* estimation with daily exchange rate data with volatility as one of the predictors. An important conclusion from Taylor (1999) is the need for a large training sample. A major difference between our approach and Taylor (1999) is that we model multiple quantiles in one step, using a single multi-output NN and we include the volatility model within our NN. A second difference is the NN architecture itself. Since volatility is persistent, adding the volatility within the NN requires more time series structure. This we accommodate by using an LSTM.

Another contribution is the sampling method for training the neural network. We apply random sampling of sequences of returns and realized variances to ensure that we include both quiet, volatile and crisis periods for estimation. The NN must have seen previous crisis events in order to recognize the onset of a new crisis, if such a new crisis can be predicted at all. The validation sample must also contain a broad mix of different market circumstances to properly evaluate the performance of the model. In that sense our method differs from standard econometric practice of using rolling windows to estimate a relatively parsimonious model. After a quiet period such a model has lost data information about the dynamics of volatility and quantiles around a crisis, and will not be able to quickly adjust *VaR* estimates. Whether a complicated model such as an LSTM trained on a long sample will do better than a simple model estimated on a rolling window, depends on the underlying process. If there is a recurring pattern in recognising quiet periods and crises, the LSTM trained on a long sample will outperform the parsimonious rolling window model. If, on the other hand, the process is subject to structural breaks, patterns from a previous crisis will not be informative on the start of a new crisis. In a world with structural breaks, characterized by 'this time is different', every new crisis has its own typical characteristics and the use of training a NN on a long data series is limited.

Nieto and Ruiz (2016) review the empirical evidence on forecasting conditional *VaR*. For conditional *VaR* they distinguish two main approaches. The first is the two-stage approach consisting of a volatility model for conditional variance plus a model for the quantiles of the scaled returns. Most applications use some form of GARCH to estimate conditional volatility. In recent years, the second approach, Realized Volatility has become more popular, as it appears to produce more accurate *VaR* forecasts. In our benchmark models we will therefore take the Corsi (2009) HAR model to estimate conditional volatility.

2 Methodology

We aim to predict volatility and Value-at-Risk (at different quantiles) at different horizons using daily data on returns and realized variance. Since *VaR* depends heavily on the volatility of a stock, we model volatility and *VaR* jointly, using a latent dynamic model with states that are common to both *VaR* as well as volatility.

2.1 Variable definitions

Let r_t be the daily close-to-close logarithmic return on a stock or stock index, and let r_{jt} be the j^{th} intraday return on day t. Intraday logarithmic returns are used to construct realized variance (RV)

$$RV_t = \sum_j r_{jt}^2. \tag{1}$$

For the intraday returns we will use a 5-minute frequency, which gives 77 observations per day. The 5-minute frequency offers a good balance between bias and variance (Liu, Patton, and Sheppard, 2015). RV is an accurate measurement of volatility, but misses the overnight returns, whereas the daily squared return r_t^2 is a very noisy measure of volatility, but covers the full day. Shephard and Sheppard (2010) advocate to use both variables jointly for modelling volatility.

Both GARCH type conditional volatility as well as realized volatility show asymmetric responses to past returns. Volatility forecasts can be improved by taking into account this leverage effect. Following Glosten, Jagannathan, and Runkle (1993) and Corsi, Audrino, and Renó (2012) we therefore add the negative part of the return,

$$RM_t = -\min(r_t, 0), \tag{2}$$

as a predictor.

We will use both RV and RM to predict volatility and VaR. Based on the evidence in Corsi (2009) we allow for a large number of lags for RV. For the asymmetry we expect a much shorter memory. All inputs are stored in the $(2 \times K)$ matrix X_t as

$$\begin{aligned} X_t &= \begin{pmatrix} RV_{t-K+1} & RV_{t-K+2} & \dots & RV_{t-10} & RV_{t-9} & \dots & RV_t \\ 0 & 0 & \dots & 0 & RM_{t-9} & \dots & RM_t \end{pmatrix} \\ &= \begin{pmatrix} x_{t-K+1} & \dots & x_{t-1} & x_t \end{pmatrix} \end{aligned} \tag{3}$$

The second line partitions X_t as a list of (2×1) input vectors x_{t-j} to be processed sequentially starting at the longest lag. In all models we set $K = 60$, but only use the most recent 10 lags of RM.

Outputs of the model are stored in a vector $\hat{Y}_t$ consisting of three types of variables: conditional variance, expected daytime realized variance, and conditional VaR. A variance forecast is defined as

$$\hat{y}^V_{t,L} = \mathrm{E}\left[y^2_{t+L} \middle| X_t \right] \tag{4}$$

for $y^2_{t+L} = \frac{1}{L}\sum_{\ell=1}^{L} r^2_{t+\ell}$. This definition implicitly assumes a zero expected return for short horizons of a few days. VaR outputs $\hat{y}^q_{t,L}$ are defined by the quantiles,

$$\Pr\left[y_{t+L} < \hat{y}^q_{t,L} \middle| X_t \right] = q, \tag{5}$$

for different quantiles q. In practice we only consider the 1%, 5%, and 10% VaR. Finally, for the realised variance we use the outputs

$$\hat{y}^{RV}_{t,L} = \mathrm{E}\left[y^{RV}_{t+L} \middle| X_t \right], \tag{6}$$

with $y^{RV}_{t+L} = \frac{1}{L}\sum_{\ell=1}^{L} RV_{t+\ell}$.

Putting everything together, for the one-step-ahead output we have the (5×1) vector

$$\hat{y}_{t,1} = \begin{pmatrix} \hat{y}^{RV}_{t,1} & \hat{y}^{V}_{t,1} & \hat{y}^{1\%}_{t,1} & \hat{y}^{5\%}_{t,1} & \hat{y}^{10\%}_{t,1} \end{pmatrix}.$$

In models where we only consider the one-step forecasts we use the shorthand notation $\hat{y}_t = \hat{y}_{t,1}$. For the multi-period predictions the output vector is augmented to (subsets of)

$$\hat{Y}_t = \left(\hat{y}_{t,1}, \ldots, \hat{y}_{t,L} \right),$$

where the number of elements in $\hat{Y}_t$ equals $N_y = 5L$.

2.2 Long Short-Term Memory model

A neural network (NN) takes a vector of inputs X_t to produce a vector of outputs $\hat{Y}_t$. With a single hidden layer the inputs are first transformed to a vector of hidden neurons H_t, and then through another transformation to the final outputs,

$$X_t \longrightarrow H_t \longrightarrow \hat{Y}_t,$$

where the transformation from $X \to H$ consists of highly nonlinear regression functions. When the dimension of the input vector is large but structured, as in our case with X_t consisting of K lags x_{t-j}, we may not want to allow for all possible interactions among the inputs and hidden neurons. A recurrent neural network (RNN) adds more structure by processing the elements x_{t-j} in X_t such that the hidden layer neurons H_t go through a series of sequential updates. The hidden layer is initialized at $h_{t-K} = 0$, and updated recursively as

$$h_\tau = \psi(h_{\tau-1}, x_\tau), \qquad \tau = t - K + 1, \ldots, t. \tag{7}$$

At the final step the construction is terminated by $H_t = h_t$. In the model the hidden states are re-initialized at zero for every t.[1] The final output is produced by the output

[1] Because the hidden layer is re-initialized every period, notation is somewhat ambiguous. In a stricter notation, the neurons need a double subscript $h_{\tau,t}$, with the first index running from $t - K$ to t in (7) and the second index referring to the input matrix X_t being processed. The recursion is then initialized at $h_{t-K,t} = 0$ and output at time t is a function of $h_{t,t}$. Our notation follows the literature, *e.g.* Fischer and Krauss (2018), by omitting the second t index.

layer,

$$\hat{Y}_t = \Phi_Y(H_t). \tag{8}$$

This design is shown in figure 1a. The important part of the design is that the function $\psi(\cdot)$ does not depend on τ or t. Hence the model provides a pooling of the dynamic structure and implies shared dynamics for the data. A special case of the $\psi(\cdot)$ function is the linear model,

$$h_\tau = \boldsymbol{A} h_{\tau-1} + \boldsymbol{B} x_\tau, \tag{9}$$

where the hidden layer h_t has N_h elements (neurons), and matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are the same for each time step. Solving (9) for the final value $H_t = h_t$ then gives

$$H_t = \sum_{j=0}^{K-1} \boldsymbol{A}^j \boldsymbol{B} x_{t-j}. \tag{10}$$

In other words, the hidden neurons are moving sums with weights $\boldsymbol{A}^j$ of linear combinations of the inputs $\boldsymbol{B} x_{t-j}$. For suitable choices of $\boldsymbol{A}$ and $\boldsymbol{B}$ the neurons may then represent slow and fast moving averages of past realized variance, similar in spirit to the Corsi (2009) HAR model, plus a few asymmetry terms as in the A-HAR model.

A much more general specification is the Long-Short-Term Memory (LSTM) model, which allows much more flexibility in the processing of the inputs. An LSTM is a special form of an RNN. The LSTM networks considered in this paper are composed of three layers: an input, a (hidden) LSTM and an output layer. A graphical representation for a single time step in the recursion (7) is shown in figure 1b. Our specification closely follows the textbook treatment in Goodfellow et al. (2016) and the finance application in

Fischer and Krauss (2018). The equations for the recursion are, for $\tau = t - K + 1, \ldots, t$,

$$
\begin{aligned}
f_\tau &= \operatorname{sigm}\left(\boldsymbol{W}_{fx} x_t + \boldsymbol{W}_{fh} h_{\tau-1} + \boldsymbol{b}_f\right) \\
\tilde{s}_\tau &= \tanh(\boldsymbol{W}_{\tilde{s}x} x_\tau + \boldsymbol{W}_{\tilde{s}h} h_{\tau-1} + \boldsymbol{b}_{\tilde{s}}) \\
i_\tau &= \operatorname{sigm}(\boldsymbol{W}_{ix} x_\tau + \boldsymbol{W}_{ih} h_{\tau-1} + \boldsymbol{b}_i) \\
s_\tau &= f_t \odot s_{\tau-1} + i_\tau \odot \tilde{s}_\tau \\
o_\tau &= \operatorname{sigm}(\boldsymbol{W}_{ox} x_\tau + \boldsymbol{W}_{oh} h_{\tau-1} + \boldsymbol{b}_o) \\
h_\tau &= o_\tau \odot \tanh(s_\tau).
\end{aligned}
\tag{11}
$$

The hidden layer $h_{\tau-1}$ is updated in several steps. The intermediate results f_τ, $\tilde{s}_\tau$, i_τ, s_τ, o_τ are all vectors of the same length as h_τ. The $\operatorname{sigm}(\cdot)$ function is the elementwise sigmoid activation function that returns weights between 0 and 1. The weights f_t, i_t and o_t are known as the 'forget', 'input' and 'output' gates. Based on the previous state $h_{\tau-1}$ and new data x_τ they give weight to past states $s_{\tau-1}$, the current input x_τ, and its relevance for the next state h_τ. The $\tanh(\cdot)$ function produces outputs between -1 and 1. The matrices $\boldsymbol{W}_{pq}$ and vectors $\boldsymbol{b}_p$ $(p = f, \tilde{s}, i, o; q = x, h)$ contain unknown parameters.

The final state for time t is $H_t = h_t$. At the end of the recursion there is an output layer similar to (8), which we take as linear,[2]

$$
\hat{Y}_t = \boldsymbol{W}_{yh} H_t + \boldsymbol{b}_y. \tag{12}
$$

Equations for f, $\tilde{s}$, i, and o each have $N_h(N_x + N_h + 1)$ parameters in the matrices $\boldsymbol{W}_{\cdot h}$, $\boldsymbol{W}_{\cdot x}$ and vectors $\boldsymbol{b}_{\cdot}$. The output layer adds another $N_y(N_h+1)$ parameters. Lag length K of the predictors does not add to the number of parameters. In the baseline model with 10 neurons, one-step ahead predictions and 2 predictors we have $N_\theta = 4 \times 10 \times 13 + 5 \times 11 = 575$ parameters. The bulk of these parameters is associated with the dynamic structure

[2] Since volatility and VaR all positive quantities, we also used the ReLU function $\max(0, z)$ for the output layer, to avoid negative predictions. But even without such a safety valve, our models never produce any negative predictions. During estimation, non-positive predictions are heavily penalized by the QLIKE loss function defined in (13). This automatically forces the model to produce valid in-sample predictions. The trained models also produce strictly positive predictions in the test samples. We therefore report results using the simpler linear output layer. Negative predictions are a problem, though, for some of the benchmark models, especially the asymmetric HAR (see Section 2.7 below).

linking X_t to H_t.

2.3 Loss function

Since conditional mean, variance and quantiles are not directly observed, the estimation proceeds through the choice of appropriate loss functions.

For scale variables, such as a conditional variance, a popular choice is the Quasi-Likelihood (QLIKE) loss function as defined in Patton (2011):

$$\text{QLIKE}(y, \hat{y}) = \frac{y}{\hat{y}} - \ln\left(\frac{y}{\hat{y}}\right) - 1, \tag{13}$$

where y is the output and $\hat{y}$ its predictor. We will use the QLIKE loss for both the conditional variance and the realized variance. Since $y > 0$, QLIKE is only defined for $\hat{y} > 0$; hence it forces the learning algorithm to produce non-negative forecasts in the training sample.

For *VaR* we use the tilted absolute value loss function, introduced by Koenker and Bassett (1978) for estimating quantile regressions (see Gourieroux and Jasiak (2010) for applications to estimating *VaR*),

$$\text{QUANT}(y, \hat{y}) = \max\left(q(y - \hat{y}), -(1-q)(y - \hat{y})\right), \tag{14}$$

where q is the quantile of interest and $\hat{y}$ an estimator of y, as before. Choosing $q = 0.5$ is similar to optimizing on the median (or optimizing according to the Mean Absolute Deviation loss). Both QLIKE and QUANT are loss functions that solely depend on the output variable y and its predictor $\hat{y}$ and can be trained using user-defined loss functions in a neural net optimizer.

With multiple outputs we need to set weights for the loss functions of the different outputs. The overall loss is the weighted sum of the individual losses,

$$L = \sum_{j=1}^{N} w_j \sum_{t=1}^{T_0} L_{t,j}, \tag{15}$$

with $L_{t,j}$ the loss (either QLIKE or QUANT) for output $Y_{t,j}$ and w_j its weight where N is the number of outputs and T_0 is the number of training sample observations. We adopt an iterative approach to construct data-based weights. We start with equal weights $w_j = 1$. After a pre-defined number of epochs, *i.e.* number of passes through the data plus parameter updates, we compute the loss for each output and set the weights to the inverse of the estimated losses. We then continue the optimization with the estimated weights.

However, we find that the weights do not appear very important, as different weighting schemes did not deliver substantial differences. One reason is that parameters for the output layer are completely separate for each output, and can be freely optimized per output variable. Conditional on the hidden layer neurons, prediction errors for one output do not jeopardize the fit for another. Second, and one of the main motivations for our choice of outputs, is that the output variables are highly correlated. Any choice of LSTM parameters that produces more informative neurons for one output, will likely be beneficial for all outputs.

In general, the larger the weight on the loss for a particular element in Y_t, the better the model will fit that element at the cost of other elements. That cost will turn into an advantage if the outputs are highly correlated. By forcing the signal H_t to fit all the different outputs, the multiple outputs become a regularization device. The bulk of the LSTM parameters are associated with the dynamic model for H_t. Pooling these parameters to predict multiple outputs limits overfitting, since the same neurons that predict one element of the output vector must also have predictive power for all other outputs.

Outputs in our model very likely will exhibit strong correlations. For the two variances, $\hat{y}_t^V$ and $\hat{y}_t^{RV}$, the correlation is obvious. The *VaR* quantiles are also closely connected to the volatility estimates. The one-day ahead *VaR* is approximately

$$\hat{y}_t^q \approx z_q \sqrt{\hat{y}_t^{RV}}, \tag{16}$$

with z_q the q^{th} quantile of the scaled returns. We therefore expect that a model that performs well in predicting volatility, will also be a strong contender for conditional VaR.[3]

Having realized variance among both inputs as well as outputs further adds to improving the information content of the hidden layer. Because of its better signal-to-noise ratio, RV can be better predicted than squared returns. Pushing the neurons to fit RV, will also provide them with information for the other variables. Similarly, due to the strong persistence of the volatility process, longer horizon forecasts for volatility will be highly correlated with one-day ahead predictions.

2.4 Early Stopping

Given the large number of parameters some form of regularization is required to prevent overfitting. In the previous subsection we already discussed parameter sharing by pooling of the dynamic structure of the network as one regularization method. For further regularization we primarily rely on early stopping based on the loss in a validation sample.

The total number of observations is T. Based on T we set a batch-size $T_b = T/7$. We retain $T_2 = 2T_b$ observations as a test sample that will not be used for estimation or the selection of the early stopping time. The remaining data are split in a estimation sample with $T_0 = 4T_b$ observations and a validation sample of length $T_1 = T_b$.

We feed the data to the network in batches of size T_b. For each batch the optimizer updates the parameters. After each epoch, consisting of four batches, we compute the loss in the validation sample. Iterations are stopped when the validation loss is minimized. However, the optimization process is not always monotonic in its improvements of the validation loss; a local minimum may be encountered and a strict early stopping would stop the process at this local minimum. That would prevent potential improvements of the model. That is why we apply a patience parameter that 'waits' before stopping the optimization. We use a patience parameter of 50, meaning that the algorithm only stops when the model loss has not been improved for 50 epochs.

We also experimented with dropout. Dropout consists in randomly neglecting a pre-

[3] In the full LSTM the quantile z_q for the scaled return can (and will) of course be time-varying as well depending on the recent history of returns and realized variance.

defined proportion of the connections of the network in the optimization process. Adding 10% dropout did not improve the validation sample performance in our case. We therefore report results without dropout.

2.5 Implementation

The number of LSTM neurons is set by performing a sensitivity analysis. The number of neurons is initially set to 20 and adjusted upwards and downwards in order to detect whether there exists a pattern in terms of performance with respect to the number of neurons selected. The lowest and highest numbers of tested neurons are 5 and 40, respectively.

The first step consists of running the LSTM over 25 epochs (number of iterations over the whole training sample) with the MSE as loss function for the conditional variance and realized variance. We start with the MSE, and not the QLIKE, because we need a valid set of starting parameters that generate non-negative variance predictions. With random starting values some predicted variances may be negative, which will lead to infinite QLIKE loss. Once the optimization has taken place over 25 epochs, the QLIKE loss can be calculated without any problem. All results are obtained using the keras (Allaire and Chollet, 2020) and tensorflow (Allaire and Tang, 2020) packages in R, using the ADAM optimiser.

2.6 Data shuffling and samples definition

In developing a neural network model the data are separated into a training sample and a test sample. The performance of a neural network is evaluated out-of-sample using a test sample. Essentially, training data are used for the optimization only and then the testing data are used to compute fitted values and assess the performance of the model. Given the large number of parameters that neural networks often require, it is not surprising to observe a very good performance in the training sample, as many steps are used in the optimization process. However, this high number of parameters to estimate is an important issue when we use new data, as the algorithm can be prone to overfitting. To circumvent this issue, we use robustness procedures, as defined in the previous section. Early stopping requires monitoring of the loss of the model in order to be able to tell when the optimization has to stop. That is done via the validation sample. This sample is a subset of the training sample and it is not used for the parameter optimization. What happens is that the remaining part of the training sample (the estimation sample) is the only one directly used in the parameter optimization. At each step, the validation sample is used on the current model and the associated loss is recorded. It is only when the optimization is finished, potentially because of the early stopping, that the model is evaluated on the test sample data.

Practically, blocks of observations are allocated to the different samples following the model designs from the previous subsection. A block of observations can be defined as the sequence of data points from $t-K+1$ to t for the inputs and from $t+1$ to $t+1+L$ for the outputs, given a forecasting horizon L. We set $K = 60$ for RV and $K = 10$ for the past negative returns. The last element of each block is always its most recent data point. We make a key distinction in the way observation blocks are allocated to each subsample. The test sample data are either randomly selected from the full sample or always taken as the last part of the entire sample, at the end of the calendar period. Once the test sample data have been identified, the remaining data points are allocated to either the estimation or validation sample randomly, for both designs.

The random block allocation is the design that is expected to perform the best in terms of forecasting power, as the optimization process of the LSTM model encounters a large variety of volatility states and the testing data are supposed to have similar properties as the training data. The non-linear nature of our model is expected to help when fitting the model to a new data set (test data). However, the benefits of randomly selecting the observations blocks are not restricted to our LSTM model. Both linear benchmarks should also benefit from this random allocation.

Models with a fixed test sample at the end of the calendar period should have more difficulties in delivering good predictions than the models with the random block allocation. This is due to the fact that all types of regimes may not be observed in the training sample with this type of sampling design, making it difficult to adapt to a new unknown situation.

2.7 Benchmark models

The first benchmark model is the Heterogeneous Autoregressive Regression (HAR) model for Realized Variance. It was introduced by Corsi (2009) and has become a standard in the literature. It is a parsimonious model that has the ability to reproduce the long-memory features of volatility. The HAR model, as specified in Corsi (2009), is

$$RV_{t+1} = \beta_0 + \beta_1 RV_t + \beta_2 RV_{5,t} + \beta_3 RV_{22,t} + \epsilon_{t+1}, \tag{17}$$

where $RV_{k,t} = k^{-1} \sum_{i=0}^{k-1} RV_{t-i}$.

In order to take into account nonlinear effects, we use Corsi, Audrino, and Renó (2012) as a basis to augment the original HAR by three input variables: the past day, past week and past month negative returns. We call this model the Asymmetric Heterogeneous Autoregressive Regression (AHAR) model:

$$RV_{t+1} = \beta_0 + \beta_1 RV_t + \beta_2 RV_{5,t} + \beta_3 RV_{22,t} + \beta_4 RM_t + \beta_5 RM_{5,t} + \beta_6 RM_{22,t} + \epsilon_{t+1}, \tag{18}$$

where

$$RM_{k,t} = -\min\left(k^{-1} \sum_{i=0}^{k-1} r_{t-i}, 0\right).$$

Both the HAR and AHAR are models where the dependent variable is the realized variance but we also use these models to estimate the daily squared returns. For that purpose, we replace RV_{t+1} by r_{t+1}^2.

The original HAR model is estimated using OLS so we also use this method here. The standard specification and estimation have a few disadvantages. The linear HAR could generate negative variance forecasts. The distribution of variance is also closer to log-normal. Corsi et al. (2012) therefore prefer a log-linear specification. For the same reasons we introduced QLIKE as the loss function in section 2.3. For comparability between the LSTM and the benchmark models we will therefore estimate them both by OLS as well as QLIKE.

As for both volatility measures, we estimate *VaR* for each benchmark model using two different methods. For the benchmark models estimated by OLS, we compute *VaR* estimates using the empirical density of scaled returns in the training sample. This method is comparable to the models developed in Brownlees and Gallo (2010). Scaled returns are defined as the returns divided by the square root of the corresponding predicted Realized Variance. The quantiles estimated in the training sample are retained for the test sample *VaR* calculation. For prediction in the test sample, (16) combines the fixed quantile with the predicted RV to forecast *VaR*.

For the alternative estimation method, *VaR* estimates are computed by a quantile regression where the dependent variable is the daily return and the regressors are the same as equations (17) and (18). This estimation method is also used in Chernozhukov and Umantsev (2001), where different regressors are used. The training sample parameter estimates are used in the test sample to make *VaR* forecasts.

In order to evaluate the forecasting performance of the LSTM model with the benchmark models, we apply Diebold-Mariano (DM) tests, as introduced by Diebold and Mariano (1995). DM tests are applied to each forecasted variable, including *VaR* forecasts where we use the tilted absolute value loss function QUANT from (14) as the loss function in the test.

3 Application to the S&P500

We use 5-minute intraday data on SPY, an ETF tracking the S&P500 index, obtained from TickWrite for the period Nov-1995 to Nov-2015. We augment our sample by the realized volatility estimates provided by the Oxford Man Library and daily open/close prices from YahooFinance for the S&P500 index for the period Nov-2015 to Mar-2020. The combined sample covers 25 years, 6265 observations, where market conditions have been very diverse. The data show two distinct huge outbursts in volatility (2009, 2020) plus a slow but noisy cycle. Figures 2 and 3 show the evolution of the daily returns and annualized realized volatility, respectively. The levels in figure 3 are similar to other studies (Maheu and McCurdy, 2011; Patton and Sheppard, 2009). Table 1 shows summary statistics for the variables in our models.

	Mean	StDev	Skewness	Kurtosis	Min	Max
RV	1.092	2.413	10.927	187.665	0.012	62.927
Returns	0.030	1.224	-0.372	12.889	-12.670	10.883
Squared returns	1.498	5.156	13.859	290.140	0	160.536
RM	-0.393	0.789	-4.172	33.542	-12.670	0

Returns are in units of percent per day. RV and RM are calculated according to (1) and (2). Returns and squared returns are for close-to-close daily returns. Skewness and Kurtosis are the third and fourth standardized moments.

Table 1: Summary statistics

3.1 Random training and test samples

Results in this subsection are obtained when observations are randomly allocated to training and test samples. The model output is the vector $\hat{y}_{t,k}$, for each horizon k, containing RV, squared returns and three levels of VaR. The LSTM layer contains 10 neurons, which means that we have 575 parameters to estimate in total (520 for the LSTM layer and 55 for the layer with the five outputs). The regularization in the model is achieved by the shared LSTM layer parameters among all output variables. Only 11 parameters in the output layer are specific for a particular output variable. Diebold-Mariano (DM) tests are performed in order to assess the difference in performance between LSTM and other models for each horizon and output variable. We apply a Newey-West correction for horizons greater than one day on reordered data, taking into account the number of missing days between ordered test observations.

Table 2 shows the values of the loss functions for all output variables, horizons and models. The first output variable is RV. For all horizons, LSTM produces the smallest QLIKE among all models considered. Standard HAR and AHAR models are estimated by OLS under MSE loss. It cannot then be expected that these models perform the best when considering their average QLIKE loss. For a fair comparison we re-estimate

		QLIKE		QUANT		
horizon	model	RV	r^2	$VaR_{1\%}$	$VaR_{5\%}$	$VaR_{10\%}$
1 day						
	LSTM	**0.152**	1.425	0.037	**0.123**	**0.201**
	HAR	0.199	1.443	**0.036**	0.124	0.201
	AHAR	1.291	9628	0.058	0.145	0.216
	HAR(QL)	0.164	**1.420**	0.041	0.127	0.204
	AHAR(QL)	0.157	1.421	0.038	0.126	0.203
5 days						
	LSTM	**0.141**	**0.355**	0.092	0.298	0.470
	HAR	0.186	0.373	0.090	0.292	**0.468**
	AHAR	0.178	2.995	0.094	0.301	0.474
	HAR(QL)	0.147	0.366	**0.090**	**0.292**	0.470
	AHAR(QL)	0.142	0.360	0.090	0.300	0.474
10 days						
	LSTM	**0.170**	**0.296**	**0.134**	**0.397**	**0.626**
	HAR	0.221	0.318	0.139	0.415	0.641
	AHAR	0.202	0.483	0.135	0.418	0.643
	HAR(QL)	0.191	0.313	0.143	0.425	0.652
	AHAR(QL)	0.187	0.303	0.138	0.421	0.652
20 days						
	LSTM	**0.192**	**0.266**	**0.148**	**0.511**	**0.832**
	HAR	0.247	0.283	0.175	0.546	0.860
	AHAR	0.234	0.300	0.174	0.550	0.863
	HAR(QL)	0.205	0.277	0.177	0.552	0.864
	AHAR(QL)	0.203	0.273	0.179	0.561	0.866

Results are for a random test sample consisting of 1790 observations. The QLIKE and QUANT columns are computed as the average over the loss functions as given in (13) and (14). Losses are computed for all horizons and different models. HAR and AHAR models are computed with OLS while HAR(QL) and AHAR(QL) are optimized on the QLIKE loss. For HAR and AHAR the *VaR* estimates are based on the empirical density of the standardized returns. The forecasting performance of the LSTM model is compared to the other models via Diebold-Mariano (DM) tests. **Light red** cells signal outperformance of the LSTM. **Darker red** cells correspond to relatively higher DM statistics.

Table 2: Average losses for randomly selected test data

these models under the same QLIKE loss as the LSTM model. Almost by construction, the QLIKE-optimized versions of HAR and AHAR models systematically outperform the standard models when evaluated under QLIKE loss. This difference is consistent over all horizons considered. The HAR(QL) and AHAR(QL) models are also much closer to the LSTM model in terms of QLIKE loss. These models are able to gain performance compared to the standard HAR and AHAR in periods of low volatility, as shown in figure 4.

The strong performance of the LSTM for realized volatility is mostly due to sample periods when volatility is low, as was already observed from figure 4. Figure 5 shows the cumulative sum of the loss difference between HAR and LSTM in red and HAR(QL) and LSTM in blue, where observations are ordered on the observed RV. HAR is then able to gain on LSTM marginally on the rest of the sample. For HAR(QL), however, the graph shows that LSTM is gaining consistently in all economic environments. These observations are valid for all horizons considered. DM tests show that the performance of the LSTM versus the other models is very good in three quarters of the tests performed, as can be observed from the number of green cells in the first column of table 2.

In comparison to RV, squared returns are notoriously more difficult to forecast, given their low signal to noise ratio. This is confirmed by figure 6, where we see that the forecasts are far from the volatility measure for several observations. The very poor score of AHAR for squared returns predictions at the one-day horizon is to be related to the bottom observations on figure 6b. For these observations, AHAR generates negative forecasts, which are not possible in reality, so we replace them by the smallest non-zero squared return observed in the training sample. This correction is not necessary for the LSTM and for the HAR models estimated with QLIKE loss. The performance of all models for forecasting this noisy variable improves as the forecasting horizon increases as seen in Table 2. The LSTM model produces the smallest QLIKE loss for all horizons, except at the one day horizon, where it is just behind HAR(QL) and AHAR. DM tests show that the relative underperformance of LSTM at the one day horizon is minor and that LSTM is edging the other models especially at the ten days horizon.

The last output variables are the three levels of *VaR*. The relative performance of the models does not generally differ with the *VaR* level chosen. However, contrary to what we observe for the other two output variables, the linear models optimized on QLIKE do not outperform their OLS counterparts when forecasting *VaR*. The LSTM, once again, performs well, especially at the 10 and 20 days horizons, where its model loss is the lowest for all levels. At the 1 day horizon, only the HAR model has a smaller loss than the LSTM. The 5 days horizon results are not in line with the other horizons considered. Here, HAR(QL) and HAR perform the best.

Figure 7 shows the cumulative sum of loss differences between LSTM and both benchmark models obtained via OLS, for the one day horizon. The general pattern does not differ between models and *VaR* levels. In all cases, the LSTM generates better *VaR* forecasts when returns are the smallest (low negative returns), which is when *VaR* forecasts need to be particularly effective. The outperformance region in the lower tail expands as the *VaR* level increases, and is always followed by a roughly equivalent interval of LSTM underperformance. LSTM then generates consistently better forecasts for approximately half of the sample, but loses a bit of its advantage when returns are high. DM tests show that when LSTM is not the best performing model, it is not lagging far behind. On the other hand, when LSTM is the leading model, is significantly leading most other models, especially at the 20 days horizon. The *VaR* at the 10% level appear to be closer to each other than for the remaining levels.

Overall, the results for this sampling technique confirm that our LSTM model is able to take advantage of the regularization imposed by the model structure and outperform well-established models such as HAR and AHAR, especially for predicting realized variance and squared returns, but also for value-at-risk.

3.2 Fixed test sample at the end of the calendar period

In this subsection, we present the results of the design where the test sample is chosen as the last 1790 observations of the entire sample, slightly more than the last seven years (Aug-2013 to Mar-2021). Training is performed on the rest of the dataset. Observations in the training set are randomly allocated to either the estimation or validation sample. Table 3 provides the values of the loss functions for all output variables, models and horizons considered.

Losses for the volatility forecasts are larger, and at longer horizons also substantially larger, than with random test samples. This either means that the more recent volatitilites are harder to predict, or that the volatility process has changed such that past histories

horizon	model	QLIKE RV	QLIKE r^2	QUANT $VaR_{1\%}$	QUANT $VaR_{5\%}$	QUANT $VaR_{10\%}$
1 day						
	LSTM	0.239	1.562	0.039	**0.114**	**0.178**
	HAR	0.296	140.0	**0.037**	0.116	0.180
	AHAR	3.753	1941	0.043	0.122	0.186
	HAR (QL)	0.228	1.548	0.039	0.120	0.185
	AHAR (QL)	**0.219**	**1.520**	0.040	0.120	0.185
5 days						
	LSTM	0.253	0.501	0.112	0.281	0.425
	HAR	0.316	0.571	0.100	0.278	**0.423**
	AHAR	0.257	6.673	0.105	0.278	0.424
	HAR (QL)	0.242	0.489	**0.094**	**0.278**	0.433
	AHAR (QL)	**0.235**	**0.487**	0.098	0.279	0.436
10 days						
	LSTM	0.316	0.457	0.177	0.429	0.625
	HAR	0.361	0.497	0.153	0.417	0.626
	AHAR	0.320	0.918	**0.146**	**0.415**	**0.622**
	HAR (QL)	0.297	0.438	0.159	0.425	0.641
	AHAR (QL)	**0.295**	**0.434**	0.154	0.436	0.641
20 days						
	LSTM	0.442	0.616	0.272	0.608	**0.896**
	HAR	0.449	0.602	0.251	**0.600**	0.896
	AHAR	0.430	0.725	**0.249**	0.601	0.897
	HAR (QL)	0.420	0.572	0.251	0.601	0.907
	AHAR (QL)	**0.418**	**0.570**	0.249	0.601	0.912

Results are for a fixed test sample at the end of the calendar period consisting of 1790 observations. **Light blue** cells indicate benchmark models that outperform the LSTM. **Darker blue** cells indicate higher DM statistics. See table 2 for further notes.

Table 3: Average losses for fixed test sample at the end of the calendar period

are less relevant for forecasting. The *VaR* losses are, however, very similar and even marginally lower to what we have seen for the random test samples. In that respect the final calendar period seems representative for the entire sample. We are most interested in the relative performance of one model against the other models. In that regard, the LSTM performance has deteriorated for all horizons. In some cases the benchmark models do significantly better than the LSTM, especially the AHAR model.

Similar to the results in the last subsection, the linear models optimized on QLIKE perform better than their MSE-optimized counterparts when forecasting realized volatility and squared returns but the opposite is true most of the time for the *VaR* forecasts. For realized volatility, the AHAR(QL) model is the one with the smallest average loss. Looking at this metric, LSTM lies in between the QLIKE-optimized and MSE-optimized models. The results follow the same patterns over all horizons considered, but the difference between the models seems to fade away as the forecast horizon increases (as shown by the DM tests outcomes). That is not specific for our model, but generally true when comparing longer forecast horizons. Figure 8 shows that HAR overestimates RV when volatility is low, while this problem does not occur with HAR(QL). This was also observed in random test samples before and is due to the way the QLIKE loss function works. QLIKE considers the relative prediction error whereas MSE minimises the absolute errors. On the right panel of figure 8 the performance of LSTM and HAR(QL) seems more even. Figure 9 indicates that the LSTM outperformance compared to HAR is driven by the first half of the test sample (when observations ordered on observed RV), while HAR(QL) outperformance on LSTM comes from the first quarter (performance is equivalent for the rest of the sample).

Regarding *VaR* forecasts, models appear very similar. The only recurring pattern is that QLIKE-optimized models perform worse than the corresponding MSE-optimized models (except for the 5 days horizon). LSTM performs average, significantly outperforming the other models only at the one day horizon for the 5- and 10-% levels. Figure 10 shows that the LSTM outperformance is driven by specific parts of the test sample. At the 1% level, the very first observations are better predicted by HAR and HAR(QL), which penalizes the overall performance of the model. For the other *VaR* levels, LSTM pulls its accuracy from the highly negative returns, namely when *VaR* specifically matters. When compared with HAR(QL), LSTM loses a bit of forecasting ability after the distress period to consitently regain a significant advantage over the rest of the sample. However, it is not able to reproduce the same pattern with the comparison with HAR, as it consistently reduces its advantage gained during the distressed period over the rest of the sample.

4 Forecasts combination

For a deeper understanding of the differences between the LSTM and HAR (AHAR) models, we compare their predictions. Both the LSTM and AHAR predict realized volatility by a linear combination of several predictors. For AHAR these are six predetermined variables, while for LSTM we use ten latent variables whose construction is completely data driven.

Predictors for the LSTM model have been defined as the vector H_t in (12). The predictors for AHAR have been defined in (18), which we denote by the vector G_t. The first question we ask is how closely the two sets of predictors are related. For this purpose, figures 11 and 12 show the canonical correlations between G_t and H_t, both for training and test data. The first canonical correlation between G_t and H_t is almost equal to one. In other words there exist linear combinations of the sets of predictors that are almost perfectly correlated. For a series such as RV this is just the level effect of volatility, and in line with the unit root like behavior of RV. The second and third canonical correlations are also high (around 0.8), indicating a close connection between the two types of models. The further canonical correlations are much lower, indicating differences between the potential predictions of the two models.

The high correlations support both LSTM and HAR. For the HAR it means that the different moving averages (daily, weekly, monthly) are consistent with what is obtained from a data driven algorithm attempting to find the best predictors. At the same time it shows the power of the LSTM in being able to construct high quality predictors. A second observation on the canonical correlations is the stability of the pattern between test and training samples.

The results for the random sampling versus fixed final test period are similar, apart from slightly lower canonical correlations in the latter case. When the LSTM is fed with data excluding the last seven years, it has seen less diverse histories. Consistent with what we observed in the loss functions, the predetermined predictors of the HAR (AHAR) perform differently under these circumstances.

The predictions of the two models are highly, but not perfectly, correlated. A natural question is therefore whether a forecast combination adds value. Given the prediction by each model, LSTM versus AHAR(QL), we run the encompassing regression

$$Y_{t+L} = c_0 + c_1 \hat{y}_t^{LSTM} + c_2 \hat{y}_t^{AHAR(QL)} + v_{t+L}, \tag{19}$$

where Y_{t+L} is any of the five output variables defined in (4)–(6), *i.e.* RV, r^2, VaR(1%, 5%, 10%), at horizon L.

Values of $c_0 = c_1 = 0$ and $c_2 = 1$ imply a perfect forecast of the AHAR model while $c_0 = c_2 = 0$ and $c_1 = 1$ imply a perfect forecast of LSTM. Models are estimated with the same loss functions (QLIKE, QUANT) as before as a generalized linear model. By construction, since the encompassing models nest the LSTM and AHAR(QL), the losses must be less than those reported in tables 2 and 3 for the different sampling schemes.

Table 4 shows that the forecast combinations for the randomly selected test data only marginally reduce the prediction loss for all outputs at all horizons. Consistent with the earlier results for the Diebold-Mariano tests, the coefficient for the LSTM predictions is in most cases larger than the weight of the AHAR(QL) predictions. In many cases, except for r^2, the weight for AHAR(QL) is not significantly different from zero. This is another indication that the LSTM model generally has better predictive power.

In table 5, with the fixed calendar time test sample, the AHAR(QL) performs relatively better. Losses are now sometimes substantially below the minimum of LSTM and AHAR(QL) in table 3, indicating a potential for forecast combination. Whether that will really work out-of-sample is not clear yet, since the weights in the encompassing regressions have been estimated ex-post on the test sample data themselves. Coefficients for the AHAR(QL) are now higher than the ones for the LSTM, except for the *VaR* forecasts, where they are mostly either insignificant or lower than the LSTM ones.

As a final analysis of the predictions we look at the relations among the neurons in the LSTM. We chose the 10 neurons without much experimentation, and definitely without using the number of latent neurons as a parameter in the optimisation. In figures 13 and 14 we show network graphs for the time series of the neurons in the test sample, for both sampling schemes. Each connection in the graph denotes a partial correlation between two neurons. The width of the vertex indicates the magnitude of the correlation, with 0.3 (in absolute terms) used as threshold. For the random test sample, the neurons for the one-day ahead predictions have a very clear structure. Neuron 1 is different from all others, while neurons (2, 6, 7), (3, 4, 8, 10) and (5, 9) are clusters of similar neurons. This can be taken as some indication that four predictors may have been enough for this dataset and that each cluster contributes to the overall predictive power of the model. The same intuition can be used to interpret the neurons for the one-day ahead predictions in the other sampling scheme. There it seems that five predictors could have been sufficient. For the 20-days horizon, there are many more connections and it is no longer possible to separate sets of correlated neurons. The lack of separation between neurons, in terms of their partial correlation, implies that the relationship between the inputs, neurons and outcome variables are potentially more nonlinear and complex for longer horizons.

$y = c_0 + c_1\hat{y}^{LSTM} + c_2\hat{y}^{AHAR(QL)}$					
Estimates	RV	r^2	$VaR_{1\%}$	$VaR_{5\%}$	$VaR_{10\%}$
1 day					
c_0	-0.006	0.046	1.430	-0.242	-0.094
	(0.007)	(0.040)	(0.879)	(0.204)	(0.143)
c_1	0.797	0.395	0.737	1.043	0.834
	(0.125)	(0.184)	(0.513)	(0.406)	(0.230)
c_2	0.215	0.541	0.827	-0.177	0.080
	(0.114)	(0.170)	(0.750)	(0.443)	(0.296)
Loss	0.151	1.413	0.036	0.123	0.200
5 days					
c_0	0.007	-0.015	1.689	0.111	0.294
	(0.008)	(0.024)	(2.032)	(0.796)	(0.557)
c_1	0.607	0.677	0.972	0.679	0.742
	(0.078)	(0.108)	(0.535)	(0.323)	(0.299)
c_2	0.386	0.361	0.355	0.405	0.341
	(0.074)	(0.101)	(0.469)	(0.423)	(0.403)
Loss	0.136	0.350	0.088	0.295	0.469
10 days					
c_0	0.015	0.051	-3.675	-1.591	-0.846
	(0.014)	(0.039)	(3.333)	(0.634)	(0.440)
c_1	0.940	0.735	1.149	1.344	1.244
	(0.085)	(0.128)	(0.848)	(0.134)	(0.203)
c_2	0.033	0.227	-0.369	-0.663	-0.496
	(0.080)	(0.123)	(1.085)	(0.234)	(0.194)
Loss	0.170	0.292	0.127	0.392	0.620
20 days					
c_0	0.035	0.065	-0.253	-0.349	-0.611
	(0.017)	(0.036)	(3.669)	(0.590)	(1.065)
c_1	0.744	0.652	0.893	0.787	0.675
	(0.128)	(0.155)	(0.290)	(0.116)	(0.129)
c_2	0.199	0.271	0.080	0.144	0.163
	(0.120)	(0.150)	(0.284)	(0.028)	(0.236)
Loss	0.188	0.260	0.147	0.506	0.825
Note:	Standard errors in parentheses				

Table 4: Encompassing test sample regressions table, random training and test samples

$y = c_0 + c_1 \hat{y}^{LSTM} + c_2 \hat{y}^{AHAR(QL)}$					
Estimates	RV	r^2	$VaR_{1\%}$	$VaR_{5\%}$	$VaR_{10\%}$
1 day					
c_0	-0.010	-0.165	0.872	1.139	0.245
	(0.008)	(0.029)	(0.492)	(0.617)	(0.091)
c_1	0.219	-0.135	1.275	1.137	1.198
	(0.102)	(0.196)	(0.442)	(0.419)	(0.178)
c_2	0.736	1.556	0.232	0.634	0.044
	(0.100)	(0.248)	(0.261)	(0.738)	(0.081)
Loss	0.216	1.491	0.036	0.112	0.177
5 days					
c_0	0.007	-0.101	-1.823	1.632	0.806
	(0.011)	(0.024)	(2.905)	(0.632)	(0.988)
c_1	0.090	0.179	0.376	0.966	1.058
	(0.142)	(0.188)	(1.935)	(0.333)	(0.418)
c_2	0.871	1.204	0.495	0.536	0.208
	(0.144)	(0.218)	(1.808)	(0.169)	(0.647)
Loss	0.235	0.467	0.092	0.273	0.422
10 days					
c_0	0.060	-0.058	-0.274	-0.408	-0.563
	(0.015)	(0.039)	(5.074)	(2.763)	(1.043)
c_1	-0.302	-0.053	-0.112	1.347	1.541
	(0.128)	(0.164)	(1.125)	(0.564)	(0.455)
c_2	1.202	1.331	1.462	-0.264	-0.716
	(0.148)	(0.198)	(0.444)	(1.014)	(0.516)
Loss	0.289	0.421	0.132	0.424	0.624
20 days					
c_0	0.121	0.132	-9.678	-0.532	-6.225
	(0.040)	(0.090)	(15.674)	(2.595)	(0.468)
c_1	-0.037	-0.290	-0.692	0.048	0.771
	(0.276)	(0.275)	(1.557)	(0.140)	(0.278)
c_2	0.848	1.391	0.657	0.794	-1.198
	(0.293)	(0.324)	(0.599)	(0.540)	(0.150)
Loss	0.403	0.522	0.245	0.597	0.870
Note:	Standard errors in parentheses				

Table 5: Encompassing test sample regressions table, fixed test sample at the end of the calendar period

5 Conclusion

We develop an LSTM neural network model for the joint prediction of conditional variance of daily returns and Value-at-Risk at different horizons. We do so by pooling the LSTM dynamic structure for multiple outputs. The pooling acts as a regularizer that benefits out-of-sample forecasts. Due to the relatively small number of variable-specific parameters and the non-linear structure resulting from the hidden layers, our model is similar to a generalized (non-linear) heterogenous model. In addition, the model can be easily generalized with additional output variables and horizons.

We consider two sample allocation designs to illustrate the forecast performance of the proposed model compared to benchmarks. The results show that the sample allocation design is an important aspect in the model's performance. A fully random block allocation and random test sample, consistent with the usual machine learning framework, is the configuration that provides the best opportunity to extract the most out of the proposed model. In this sampling design, the multi-output LSTM outperforms compared to benchmarks. The second sampling design considers a fixed test sample at the end of the calendar period. In this case, the outperformance is less pronounced. In both sampling designs, our method is more efficient compared to benchmarks due to the joint modeling of multiple risk factors. We further compare the LSTM predictors (neurons) with benchmark (AHAR) model predictors in terms of their correlations and explanatory power. The LSTM predictors are not directly correlated with AHAR predictors and the former has higher explanatory power for most output measures and particularly under the random block allocation sampling design.

The outperformance of the LSTM can be due to several factors, such as a more general lag structure or additional non-linearities. This flexible framework offers potential for further investigation, in terms of the variables selected and horizons considered.

6 References

Allaire, J. and F. Chollet (2020): *keras: R Interface to 'Keras'*, r package version 2.3.0.0.

Allaire, J. and Y. Tang (2020): *tensorflow: R Interface to 'TensorFlow'*, r package version 2.2.0.

Bollerslev, T., A. Patton, and R. Quaedvlieg (2016): "Exploiting the Errors: A Simple Approach for Improved Volatility Forecasting," *Journal of Econometrics*, 192, 1–18.

Brownlees, C. and G. Gallo (2010): "Comparison of Volatility Measures: a Risk Management Perspective," *Journal of Financial Econometrics*, 8, 29–56.

Bucci, A. (2020): "Realized Volatility Forecasting with Neural Networks," *Journal of Financial Econometrics*, 18, 502–531.

Cannon, A. (2011): "Quantile regression neural networks: Implementation in R and application to precipitation downscaling," *Computers and Geosciences*, 37, 1277–1284.

Chernozhukov, V. and L. Umantsev (2001): "Conditional value-at-risk: Aspects of modeling and estimation," *Empirical Economics*, 26, 271–292.

Christensen, K., M. Siggaard, and B. Veliyev (2021): "A Machine Learning Approach to Volatility Forecasting," CREATES Research paper 2021-03.

Corsi, F. (2009): "A simple approximate long-memory model of realized volatility," *Journal of Financial Econometrics*, 7, 174–196.

Corsi, F., F. Audrino, and R. Renó (2012): "HAR Modeling for Realized Volatility Forecasting," in *Volatility Models and Their Applications*, ed. by L. Bauwens, C. Hafner, and S. Laurent, Wiley.

Diebold, F. X. and R. Mariano (1995): "Comparing Predictive Accuracy," *Journal of Business and Economics Statistics*, 13, 253–263.

Donfack, M. N. and A. Dufays (2020): "Modeling time-varying parameters using artificial neural networks: a GARCH illustration," *Studies in Nonlinear Dynamics & Econometrics.*

Engle, R. and J. Rangel (2008): "The Spline-GARCH Model for Low-Frequency

Volatility and its Global Macroeconomic Causes," *Review of Financial Studies*, 21, 1187–1222.

Fischer, T. and C. Krauss (2018): "Deep Learning with Long Short-Term Memory Networks for Financial Market Predictions," *European Journal of Operations Research*, 270, 654–669.

Glosten, L. R., R. Jagannathan, and D. E. Runkle (1993): "On the Relation between the Expected Value and the Volatility of the Nominal Excess Return on Stocks," *Journal of Finance*, 48, 1779–1801.

Goodfellow, I., Y. Bengio, and A. Courville (2016): *Deep Learning*, MIT Press.

Gourieroux, C. and J. Jasiak (2010): "Value at Risk," in *Handbook of Financial Econometrics Volume 1*, North-Holland.

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

Koenker, R. and G. Bassett (1978): "Regression Quantiles," *Econometrica*, 46, 33–50.

Liu, L. Y., A. J. Patton, and K. Sheppard (2015): "Does anything beat 5-minute RV? A comparison of realized measures across multiple asset classes," *Journal of Econometrics*, 187, 293–311.

Maheu, J. M. and T. H. McCurdy (2011): "Do high-frequency measures of volatility improve forecasts of return distributions?" *Journal of Econometrics*, 160, 69–76.

Nieto, M. and E. Ruiz (2016): "Frontiers in VaR Forecasting and Backtesting," *International Journal of Forecasting*, 32, 475–501.

Patton, A. J. (2011): "Volatility forecast comparison using imperfect volatility proxies," *Journal of Econometrics*, 160, 246–256.

Patton, A. J. and K. Sheppard (2009): "Optimal combinations of realised volatility estimators," *International Journal of Forecasting*, 25, 218–238.

Rahimikia, E. and S.-H. Poon (2020): "Machine Learning for Realised Volatility Forecasting," SSRN WP 3707796.

Shephard, N. and K. Sheppard (2010): "Realising the Future: Forecasting with

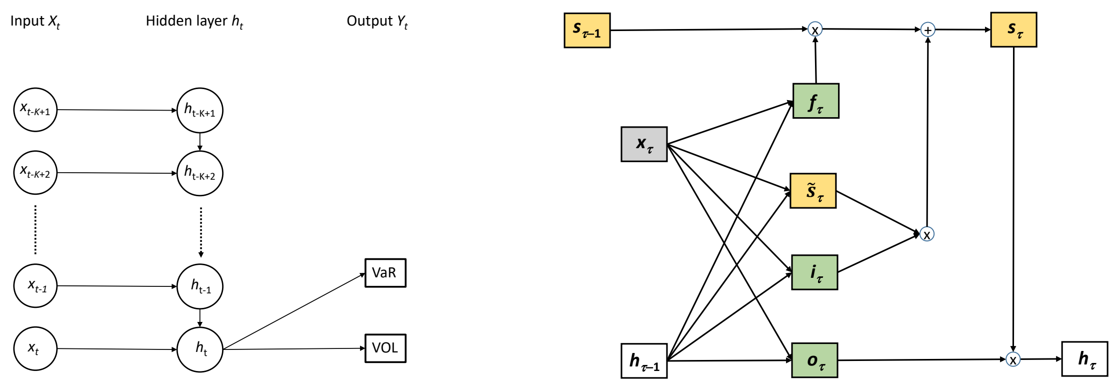
High-Frequency-Based Volatility (HEAVY) Models," *Journal of Applied Econometrics*, 25, 197–231.

Taylor, J. (1999): "A Quantile Regression Neural Network Approach to Estimating the Conditional Density of Multiperiod Returns," *Journal of Forecasting*, 19, 299–311.

White, H. (1992): "Nonparametric Estimation of Conditional Quantiles Using Neural Networks," in *Computing Science and Statistics*, ed. by C. Page and R. LePage, Springer, 190–199.

Wu, Q. and X. Yan (2019): "Capturing Deep Tail Risk via Sequential Learning of Quantile Dynamics," *Journal of Economic Dynamics and Control*, 109, 103771.

7 Figures



The left panel is a recurrent neural network with a single hidden layer in between the input $X_t = (x_{t-K+1}, \ldots, x_t)$ and the output Y_t. Elements $x_{t-\ell}$ in X_t are processed one by one, starting at the longest lag. The hidden neurons h are updated at each time-step using the next element in the input. The final vector h_t is passed to the output.
The right graph shows a single time step in the LSTM network. The current state, encapsulated in the vectors $h_{\tau-1}$ and $s_{\tau-1}$, is combined with the data input x_τ to construct the new state (h_τ, s_τ). The intermediate transformations are referred to as 'forget gate' f_t, 'input gate' i_t and 'output gate' o_t. The circled '$\times$' operators denote elementwise multiplication. The circled '$+$' denotes addition. All other arrows are nonlinear functions. Again, the final vector h_t is passed to the output Y_t.

Figure 1: Recurrent Neural Network

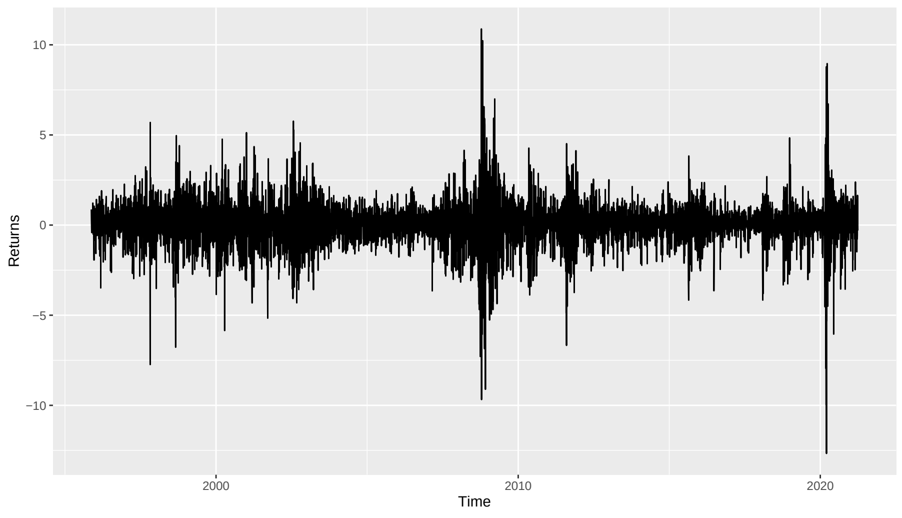


Figure 2: Daily returns

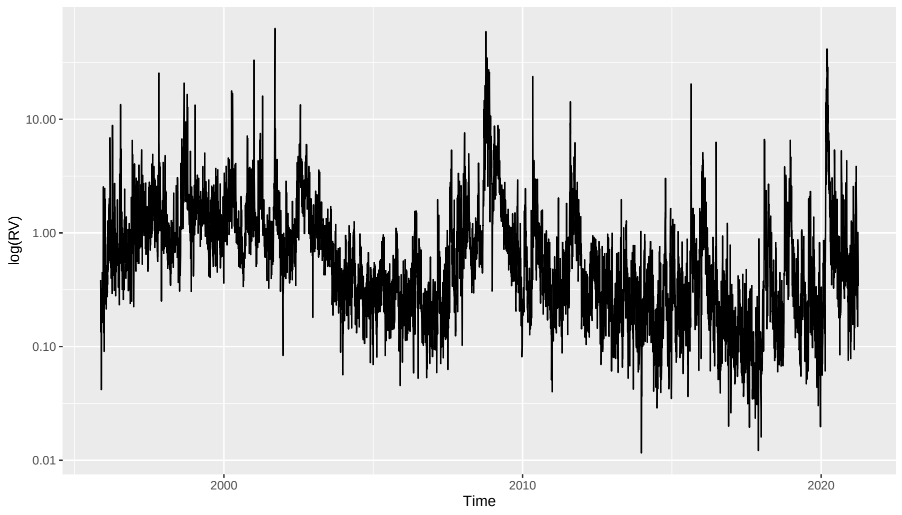
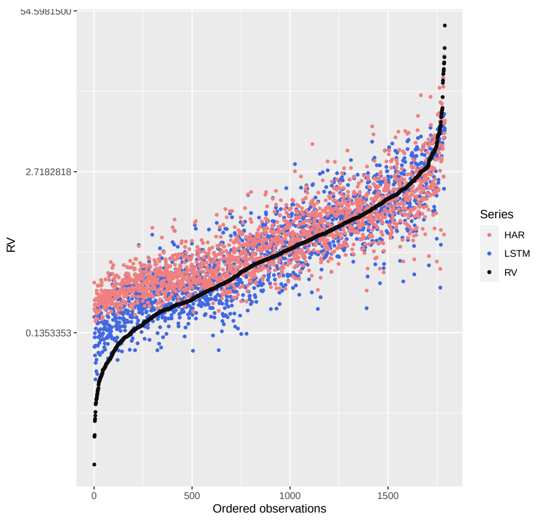
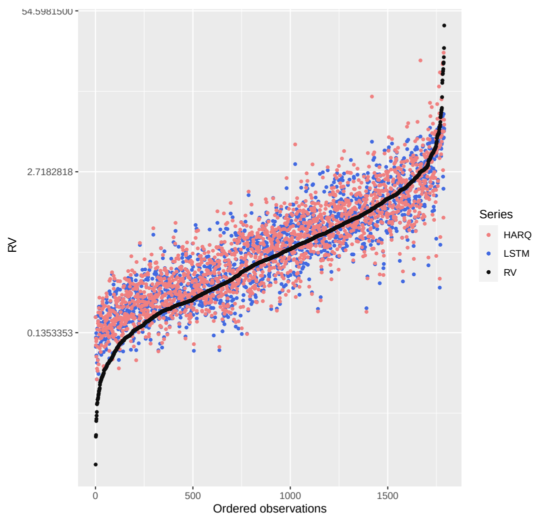


Figure 3: RV



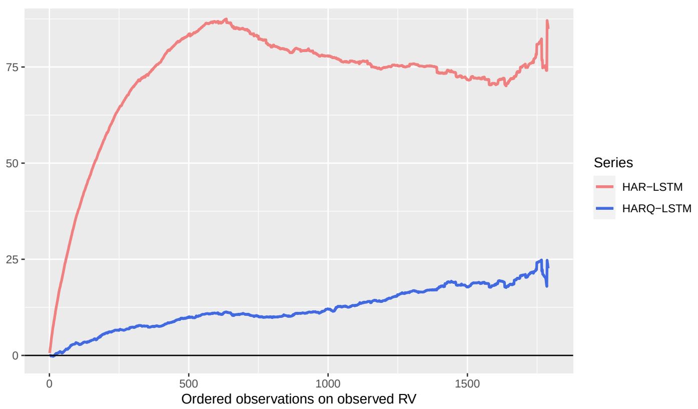
(a) RV forecasts HAR and LSTM



(b) RV forecasts HAR(QL) and LSTM

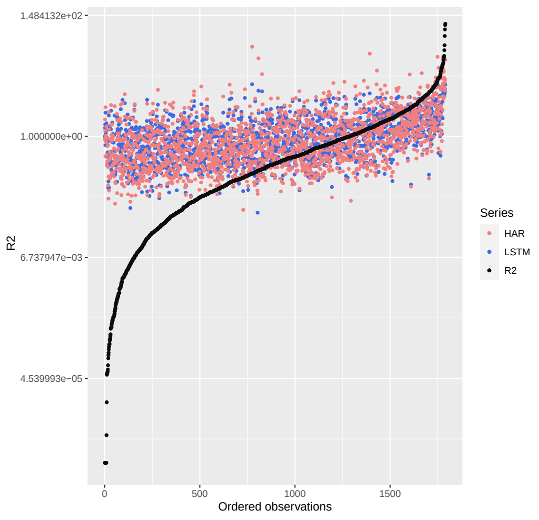
Observations are ordered on the observed RV (monotonic and upward-sloping black line). These graphs represent test sample data for the random block allocation design. The red and blue dots are the benchmark (HAR on left graph, HAR(QL) on right graph) and LSTM *RV* predictions, respectively.

Figure 4: $\hat{y}^{RV}_{t,1}$, random test data

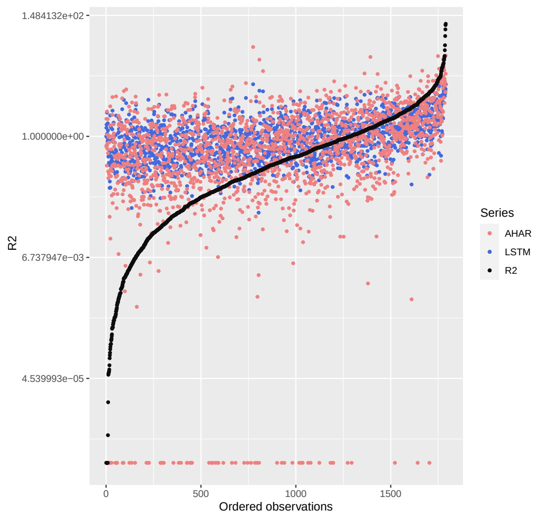


Cumulative sum of the differences in RV loss (QLIKE) between HAR(Q) and LSTM forecasts, where the data points are ordered on the observed RV.

Figure 5: $\hat{y}^{RV}_{t,1}$, random test data



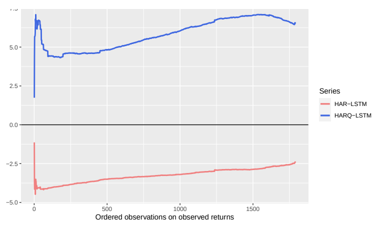
(a) r^2 forecasts HAR and LSTM



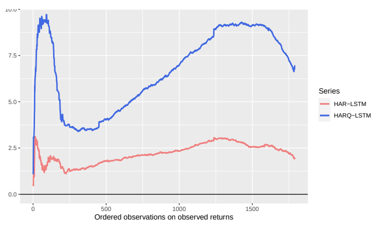
(b) r^2 forecasts AHAR and LSTM

Observations are ordered on the observed r^2 (monotonic and upward-sloping black line). These graphs represent test sample data for the random block allocation design. The black, red and blue lines are respectively the observed r^2, benchmark (HAR on left graph, AHAR on right graph) and LSTM r^2 predictions.

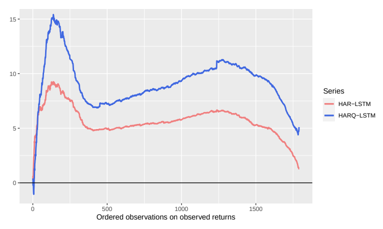
Figure 6: $\hat{y}^V_{t,1}$, random test data



(a) $VaR_{1\%}$



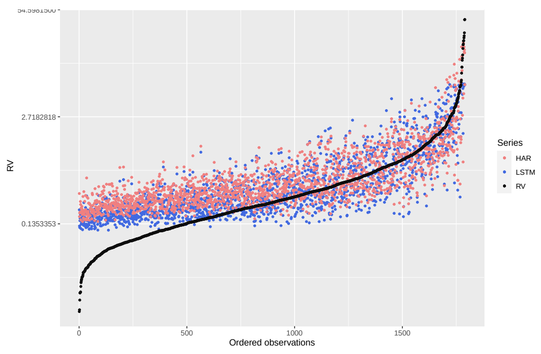
(b) $VaR_{5\%}$



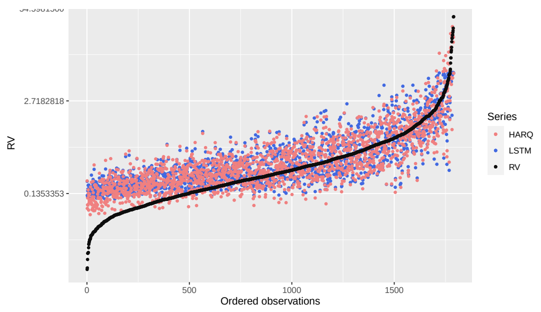
(c) $VaR_{10\%}$

Cumulative sum of loss differences between benchmark models (HAR in red and HAR(QL) in blue) and LSTM for VaR forecasts. The out-of-sample data points are ordered on the observed daily returns.

Figure 7: $\hat{y}^{\alpha\%}_{t,1}$, random test data



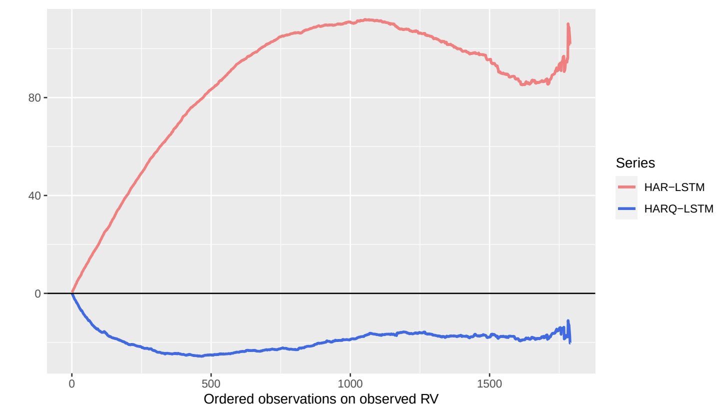
(a) RV forecasts HAR and LSTM



(b) RV forecasts HAR(QL) and LSTM

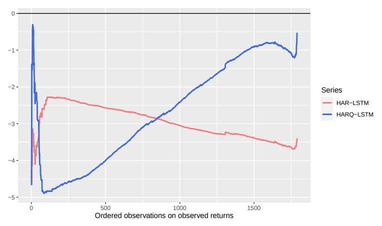
RV forecasts, where the observations are ordered on the observed RV (monotonic and upward-sloping black line). These graphs represent data from a fixed test sample at the end of the calendar period. The black, red and blue lines are respectively the observed RV, benchmark (HAR on left graph, HAR(QL) on right graph) and LSTM RV predictions.

Figure 8: $\hat{y}^{RV}_{t,1}$, fixed test sample

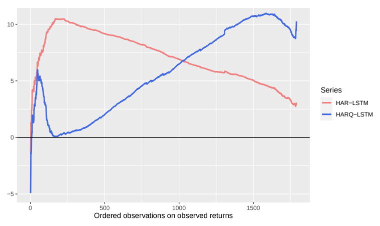


Cumulative sum of the differences in RV loss (QLIKE) between HAR(Q) and LSTM forecasts, where the data points are ordered on the observed RV.

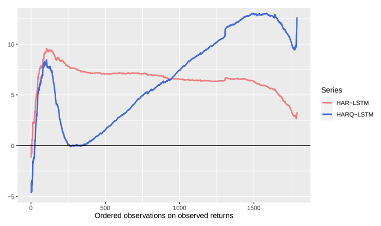
Figure 9: $\hat{y}^{RV}_{t,1}$, fixed test sample



(a) $VaR_{1\%}$



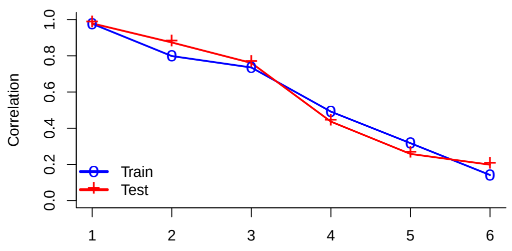
(b) $VaR_{5\%}$



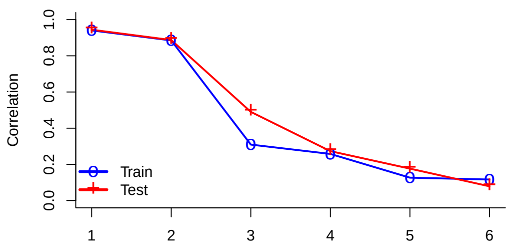
(c) $VaR_{10\%}$

Cumulative sum of loss differences between benchmark models (HAR in red and HAR(QL) in blue) and LSTM for VaR forecasts. The out-of-sample data points are ordered on the observed daily returns.

Figure 10: $\hat{y}^{\alpha\%}_{t,1}$, fixed test data

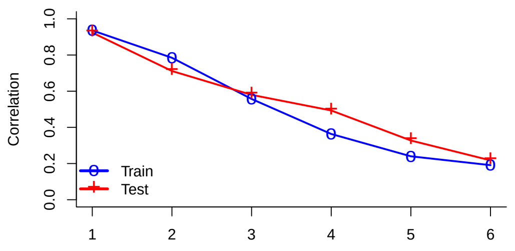


(a) 1-day horizon

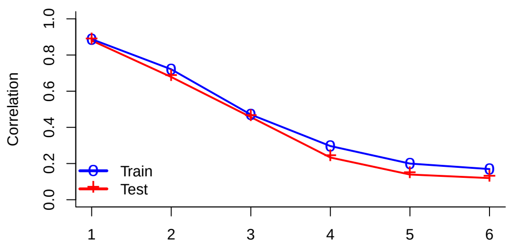


(b) 20-days horizon

Figure 11: Canonical correlations of LSTM neurons with (A)HAR components, random training and test samples

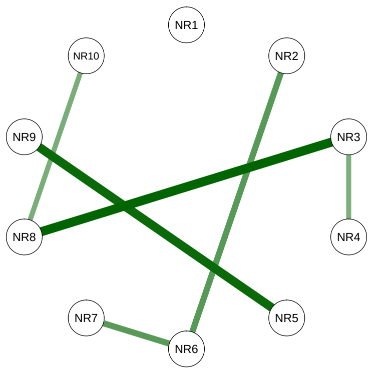


(a) 1-day horizon

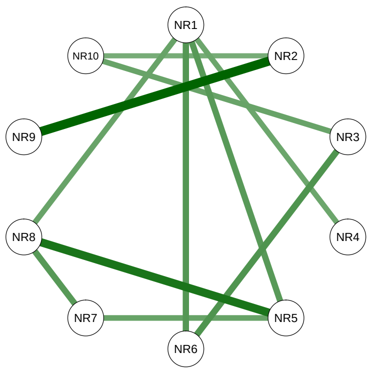


(b) 20-days horizon

Figure 12: Canonical correlations of LSTM neurons with (A)HAR components, fixed test sample at the end of the calendar period

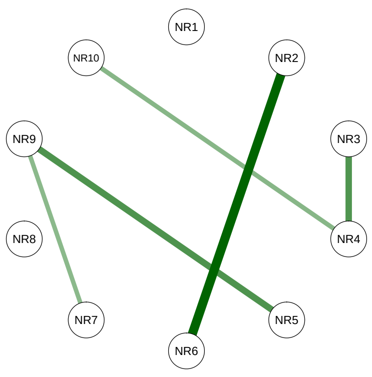


(a) 1-day horizon

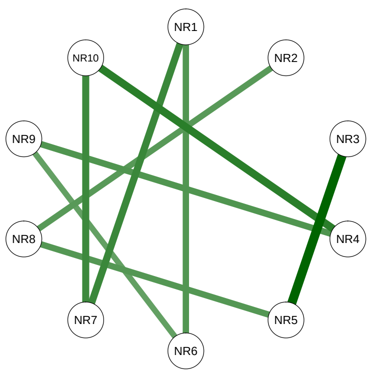


(b) 20-days horizon

Figure 13: LSTM neurons test sample partial correlations, random training and test samples



(a) 1-day horizon



(b) 20-days horizon

Figure 14: LSTM neurons test sample partial correlations, fixed test sample at the end of the calendar period