Ca' Foscari University of Venice

Master's degree programme – Second Cycle (D.M. 270/2004)

in Economics and Finance

Final Thesis

Combination of forecasts from ARIMA, Neural Networks and Hybrid models

Supervisor
Ch. Prof. Roberto Casarin

Co-supervisor
Ch. Prof. Claudio Pizzi

Graduand
Edoardo Urettini
Matriculation Number 862172

Academic Year
2020 / 2021

Abstract

The aim of the thesis is to show whether a combination of predictions from different types of models can improve predictive capabilities compared to models taken separately. Three different classes of models were used: ARIMA-GARCH models, neural networks and a hybridization between these two classes. The combination of the predictions of these different classes seeks to extract their unique capabilities in explaining a time series, going beyond the generalization provided by a single hybrid model. An application on the forecast of the VIX index is presented.

Keywords: Neural Networks, ARIMA, Model Combination, VIX

Contents

Chapter 1

Introduction

Forecasting financial time series is a long-lasting problem for which is hard to find a definitive solution. This is due to the stochastic nature of financial data, the basic complexity of the underlying process and possible changes of ”rules” in time. In their paper Abu-Mostafa and Atiya (1996) [1] considered the market as a system that takes a lot of different information while the statistical model can access only a small portion of these information. This makes the model unable to consistently predict the time series.

For this reason, this thesis uses as its starting point the famous aphorism made by Box (1976) [8] about the fact that all models are wrong. We do not think to be able to build a ”correct” model that is able to capture the variability of an entire time series, no matter how much complexity we add. But we think that we can build some ”good” models that are able to forecast something of the time series and then, if combined, we can obtain a result that overcomes the individual models.

Combination is a standard practice in the forecasting process. Starting from the paper of Bates and Granger (1969) [5] the idea of combining the results of models instead of using only the best one, grew in time with many different proposals to combine in an optimal way the forecasts. The underlying concept of this practice is that, usually, the correct data generating process is not among our candidate models and that the combination of incorrect models can improve our forecast ability.

Another way to combine models is to stack together the models themselves instead of their forecasts. In this way, the models are connected in their estimation and one can directly correct the other. In this thesis we propose a hybrid model based on two of the most common way to predict time series: the ARIMA-GARCH and the feedforward neural networks. The combination of ARIMA and neural network has already been explored by Zhang (2003) [49] in its basic form and by Tseng et al. (2002) [47] using the seasonal ARIMA. In this thesis we try to find the best ARIMA by accounting also for the heteroscedasticity of the data, estimating the parameters in a correct way. The proposed hybrid model is an ARIMA-GARCH + NN.

The forecasts obtained from the ARIMA-GARCH + NN are then combined with the forecasts provided by a single feedforward neural network and with the forecasts of a simple autoregressive model with a single lag. In this way we propose a combination of very different models trying to overcome the results of the hybrid model alone.

We propose a procedure that selects almost automatically the best ARIMA-GARCH model, the best neural network for the hybrid model and the best neural network to be used alone. This procedure is completely data-driven and tests hundreds of possible combinations, evaluating them in the training set. Particular focus was given to the risk of overfitting of the neural networks. Dropout [43] [26] and early stopping [6] regularization strategies were implemented in the estimation of the network parameters. What we propose is based exclusively on the past data of the series we want to forecast. No external regressors are used. The procedure should so be enough flexible to be applied to many kinds of univariate time series, without a single change in the code other than the input data.

This procedure is tested on the data of the CBOE Volatility Index (VIX). It is an implied volatility index [12] and it is interesting to study because it represents a sort of ”fear index” of the market about the next future. Several past papers proposed different methods to forecast the VIX index. Ahoniemi (2006) [3] used ARIMA and ARIMA-GARCH models augmented with external regressors to predict the next day direction of the VIX index. Konstantinidi et al. (2008) [30], used different models (regressions with economical variables, ARIMA, ARFIMA, VAR, PCA) to forecast the index. In these two studies, the out-sample accuracy in the prediction of the sign of the change is provided. Our approach provides significantly better accuracy than what was obtained in the two just cited papers without using external regressors. More recently Psaradellis and

Sermpini (2016) [40] applied a heterogeneous autoregressive process (HAR) combined with a genetic algorithm–support vector regression(GASV) and other hybrid models. Our hybrid model is different from what they presented. To our knowledge, this is the first application of a hybrid ARIMA-GARCH + NN to the VIX time series.

On our data, the hybrid model performed slightly better than the ARIMA-GARCH in every metric (mean squared error, mean absolute error, directional accuracy) both in-sample and out-sample. It performed also better than the single neural network almost everywhere. The combinations of the forecasts provided by the hybrid model, the single neural network and the AR(1) presented only minimal improvements on some of the metrics when compared to the hybrid model, while on other metrics they presented worsenings. Moreover, we found the simple average combination to be the most robust combination in terms of MSE being capable of minimally improving the hybrid model on both our trials. Many papers concluded the same about this way to combining forecasts [35] [13].

The thesis is structured as follows. In the second chapter the models, their estimation procedure and the selection process is presented. In the third chapter, a discussion about the combination of forecasts is provided with the different types of combinations that we will use. In the fourth chapter, we show the results of the empirical application to the VIX index.

Chapter 2

Models

In this thesis, three classes of models have been considered: ARIMA-GARCH, Feed-forward Neural Networks and a hybrid model ARIMA-GARCH + NN. The aim is to select a single model per class that can best represent its family. We want this selection to be as automatic as possible, driven mainly by the training data and with small human intervention. The idea is to create a process that can adapt to many different time series. In this chapter, the logic behind the models and the selection is presented.

2.1 ARIMA-GARCH

2.1.1 Model description

The ARIMA and GARCH models have been widely used in economic analysis and are part of the general knowledge of every professional in the sector. The Autoregressive Integrated Moving Average (ARIMA) models the conditional mean of the stochastic process using past values of the process itself (AR part) and past errors (MA part) in a linear combination of them. The ”integrated” refers to the use of differencing to make the series stationary. ARIMA(p,d,q) stays for an ARIMA with an order p AR term, an order q MA term and d differences done before the estimation. In this work the differencing term will be always 0, due to the fact that we work with series that are already stationary, so our model reduces to an ARMA model that can be written as:

$$y_t = c + \sum_{i=1}^{p} \phi_i \, y_{t-i} + \sum_{j=1}^{q} \theta_j \, \varepsilon_{t-j} + \varepsilon_t \tag{2.1}$$

where y_t is the real value of the series at time t, ε_t is the random error of the process at time t and ϕ_i and θ_j are the coefficients to be estimated. For a complete explanation look at Box, Jenkins et al. (2015) [9]. For a more concise introduction look at Adhikari and Agrawal (2013) [2]. The ARMA process can be written using the lag operator L making explicit the two polynomials of the AR and the MA part:

$$(1-\sum_{i=1}^{p}\phi_i L^i)\ (y_t-\alpha) = (1-\sum_{i=1}^{q}\theta_i L^i)\ \varepsilon_t \tag{2.2}$$

Notice that the constant α is different from the constant c found in the previous equation. The characteristic polynomials in L are now explicit. We ask for three conditions for the roots of these two polynomials [9]. Our first request is to have these two polynomials having different roots. If common roots are present, we can simplify the equation and the model can be reduced, so there is no sense in estimating a model with an higher order that is exactly equal to a lower order model. Second, we want the process to be stationary so we ask the roots of the AR polynomial to lay outside the complex unit circle. The stationarity condition. Third, we ask for invertibility, so also the roots of the MA polynomial should be outside the complex unit circle.

In this thesis, the estimation of the ARMA model is performed in R with the "rugarch" package. We are not aware of any built-in function able to test if the roots of the polynomials are following our conditions. So we code a custom function that extracts the estimated parameters of the ARMA part, compute the roots of the two polynomials and checks if there are equal roots and if the roots are all outside the unit circle. For the sake of simplicity, the function does not perform hypothesis testing considering also the uncertainty of the estimated parameters. It simply declares a model stationary and invertible if all the roots have their module greater than 1. Moreover, if there are roots of the MA and AR part that are too near in the complex plane (distance less than 0.1), the function label the model as "suspect". If the model we selected is labelled as "suspect", the researcher should check the roots plot to decide if to keep or to discard the model itself. The function is provided with further explanations in Appendix A.

The ARMA model is estimated considering the error terms ε_t to be independent and drawn from a distribution with constant variance and zero mean. Of course, it is not always the case to consider as homoscedastic the series of error, in particular when we are treating financial time series that are well known to be subject to change in their variance due to financial or economic crisis. Fitting an ARMA ignoring the heteroscedasticity of the data would put more weight on the errors of the periods of high volatility (that are higher), resulting in incorrect parameters. To solve the problem is necessary to let the conditional variance of errors to vary in time so that the distribution of errors at time t, conditional to all previous values is $\varepsilon_{t|t-1} \sim \mathcal{N}(0, \sigma_t)$. To do this we use the GARCH model, the generalized autoregressive conditional heteroskedasticity, which is a non-linear model for conditional variance. Using the notation of Danielsson (2011) [14] we can express the variance at time t conditional on all the information until time $t-1$ as:

$$\sigma_t^2 = \omega + \sum_{i=1}^{L_1} \alpha_i \, \varepsilon_{t-i}^2 + \sum_{j=1}^{L_2} \beta_j \, \sigma_{t-j}^2 \tag{2.3}$$

This is a $GARCH(L_1, L_2)$. This model is able to capture volatility clustering and persistence. To ensure a positive variance we need all the parameters to be positive and to ensure covariance stationarity we need $\sum_i \alpha_i + \sum_j \beta_j < 1$. This last constraint assure us a defined and finite unconditional variance $\sigma^2 = \frac{\omega}{1-\sum_i \alpha_i - \sum_j \beta_j}$. Given the fact that we are not really interested in the unconditional variance we will not require this constraint (look Danielsson (2011) page 39 [14]).

In the last decades, many generalizations of the GARCH model have been created and used. The R package rugarch used in this work has some of them built-in and ready for use. We select a couple of simple models to test besides the vanilla GARCH, assuring us better modelling of the conditional variance and of the whole ARIMA-GARCH model. For further details and references to the original papers, refer to ”Introduction to the rugarch package” [19].

The first model is the Exponential GARCH (eGARCH) [37]. If we define the error term to be $\varepsilon_t = \sigma_t z_t$ we can express the conditional variance as:

$$ln(\sigma_t^2) = \omega + \sum_{j=1}^{L_1} [\alpha_j \, z_{t-j} + \gamma_j \, (|z_{t-j}| - E|z_{t-j}|)] + \sum_{i=1}^{L_2} \beta_i \, ln(\sigma_{t-i}^2) \tag{2.4}$$

where the α_j is for the sign effect and the γ_j for the size effect.

The second model is the asymmetric power ARCH (apARCH) [16] expressed as:

$$\sigma_t^{\delta} = \omega + \sum_{j=1}^{L_1} \alpha_j (|\varepsilon_{t-j}| - \gamma_j\, \varepsilon_{t-j})^{\delta} + \sum_{i=1}^{L_2} \beta_i\, \sigma_{t-i}^{\delta} \tag{2.5}$$

where $\delta \in \mathbb{N}^+$ is the power parameter. This model is pretty general and allows for both leverage effect (asymmetry between positive and negative values of innovations) and power effect (taking a power of δ different from 2). It is very useful for our aims that the apARCH, remaining relatively simple, includes other submodels for some particular set of parameters. It is a superset of the Absolute Value GARCH, the GJRGARCH, the Threshold GARCH, the Nonlinear GARCH and the Log GARCH. Including the apARCH on the set of models that we use in this work allows us to have a lot of flexibility for what concerns the modelling of the volatility. We are aware of the existence of much more modern and general possibilities, but their added complexity is not needed in this work due to the fact that our main interest is to predict the conditional mean. The reason for excluding such complex models is mainly of computational nature. Our selection process relies exclusively on the data, so it tests all possible structures. Adding additional complex models for the conditional variance would imply a much higher time for selection. This does not seem necessary if the interest is the correct estimation of the conditional mean.

Finally, we have the ARIMA-GARCH (with the word GARCH we refer also to the two possible generalizations we presented) where we use the first to explain the conditional mean and the second for the conditional variance. This hybridization has been used in past works (for examples look Zhou et al. (2006) [50] or Mohammadi and Su (2010) [36]). The combination of the ARMA and standard GARCH can be written as:

$$y_t = c + \sum_{i=1}^{p} \phi_i\, y_{t-i} + \sum_{j=1}^{q} \theta_j\, \varepsilon_{t-j} + \varepsilon_t \tag{2.6}$$

where $\varepsilon_t = z_t\, \sigma_t$ and

$$\sigma_t^2 = \omega + \sum_{i=1}^{L_1} \alpha_i\, \varepsilon_{t-i}^2 + \sum_{j=1}^{L_2} \beta_j\, \sigma_{t-j}^2 \tag{2.7}$$

Where σ_t is the volatility at time t given all the information until $t-1$ and z_t are the standardized residual. Another time we remember that the GARCH is only one of the models for the conditional variance that we try during the selection of the ARIMA-GARCH. The most common conditional distribution for standardized residual is the normal distribution $\mathscr{N}(0,1)$ but other possibilities exist. We request these distributions to have 0 mean and unit variance. As suggested by Danielsson (2011) [14] one alternative to the normal is to use the Student-t distribution if the tails of the process are fatter than the ones implied by the GARCH models. In this case we need to estimate another parameter ν that describes the shape (degrees of freedom) of our conditional distribution, requiring much more data to get enough extreme values to understand the tails. The standardized t-distribution used in 'rugarch' [19] is:

$$f(\frac{x-\mu}{\sigma}) = \frac{1}{\sigma}\frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\nu-2\,\pi}\,\Gamma(\frac{\nu}{2})}(1+\frac{z^2}{\nu-2})^{-(\frac{\nu+1}{2})} \tag{2.8}$$

Where the Γ is the Gamma function.

Another distribution feasible is the Generalized Error Distribution (GED) that in its standardized form is:

$$f(\frac{x-\mu}{\sigma}) = \frac{1}{\sigma}\frac{\kappa\, e^{-0.5}|\sqrt{2^{-2/\kappa}\,\frac{\Gamma(\kappa^{-1})}{\Gamma(3\kappa^{-1})}}\;z|^{\kappa}}{\sqrt{2^{-2/\kappa}\frac{\Gamma(\kappa^{-1})}{\Gamma(3\kappa^{-1})}}2^{1+\kappa^{-1}}\Gamma(\kappa^{-1})} \tag{2.9}$$

Also in this case we have to estimate only one more parameter κ with respect to the normal case. We can do a further generalization with the skewed versions of both the Student-t and the GED. This allows for a skewness different from 0, requiring the estimation of the skew parameter in addition to the shape parameter. In the rugarch package, skewed versions of these distributions are built-in in their standardized version. The method used is the one of Fernandez and Steel (1998) [18] that for a symmetric distribution f introduced a way to make it skewed. For a skew parameter ξ we have:

$$f(z|\xi) = \frac{2}{\xi+\xi^{-1}}[f(\xi\, z)\mathbb{I}_{(-\infty,0)}(z) + f(\xi^{-1}\, z)\mathbb{I}_{[0,+\infty)}(z)] \tag{2.10}$$

Estimating the full model allowing the standardized residuals to have the skewed Student-t or the skewed Generalized Error distributions instead of restricting the possibility to the standardized normal allow us to be more general and to compute the significance of the estimated parameters in a more reliable way. Should be noted that the normal distribution is a subset of the skewed Student-t, so it will anyway be included among the possibilities.

2.1.2 Fitting and Selection

Finally, after having defined the type of the model, the orders and the conditional distribution, the estimation is executed by maximizing the log-likelihood in a single step for both the conditional variance and the conditional mean. This guarantees greater precision and efficiency when compared to a two-step estimation [19], assuring us the best parameters for the mean modelling (in which we are mainly interested). The ”hybrid” optimizer is used in this thesis. Following the ’rugarch’ documentation this optimizer tries in order the ”solnp” optimizer, the ”nlminb”, the ”gosolnp” and finally the ”nloptr”. Refer to the documentation for further details on the methods [19]. The first and main solver (”solnp”) is based on an augmented Lagrange solver [48]. All the models that have errors or warnings in the optimization process (non-convergence, non-invertible hessian etc...) are automatically discarded.

Now we have all the elements to present the selection algorithm used in this work. Here we do not follow strictly the classic Box-Jenkins [9] approach that identifies the order of the ARMA model starting from the autocorrelation and the partial autocorrelation functions. Instead, we perform a grid search that tries all the combinations of hyperparameters, followed by a selection of models with certain desirable characteristic and the construction of a final ranking based on likelihood and parsimony. The word hyperparameters includes everything that must be determined before proceeding to the estimation of the actual parameters of the model. The estimation process will estimate the parameters.

In the graph 2.1 the selection procedure is visualized. Starting with a time series stationary in mean, we try the combination of all feasible hyperparameters to find the best one. To try all the combinations we have to define the maximum order of the AR and the MA term that we want to try, the maximum orders of GARCH, the possible conditional distributions and the types of possible GARCH generalizations (apARCH, egarch, etc...). Due to computational limits, it is not easy to test all the possibilities, so it can be useful to fix some choice. In our case, we fix the maximum order of the ARMA by analyzing the type of series we are modelling and the computational power we can use; we fix the order of the GARCH to (1,1) that is the most common version [14] and should be enough for a large variety of problems; we try only the eGARCH and the apARCH with skewed Student-t and skewed Generalized error distribution for the standardized residuals. So we have to estimate 4 models for every order combination of the ARMA.

From the great number of models that we fit, as previously said, we exclude all the models that present errors or warnings during the optimization procedure. If the model passes this test, we look at all the parameters and if there are non-significant ones ($p-value >= 0.05$) we fix the parameter with the largest p-value to 0 and we refit the same model until we have all the parameters significant. We did this for all the combinations of hyperparameters that we choose. Look Appendix A to find the function that checks the significance of the parameters. Then, from all the models that we have, we exclude all of those which presents roots of the AR and MA polynomials that are inside the unit circle (non-stationarity, non-invertibility) and all models with common roots. Finally, we select the model looking at the AIC and BIC. The Akaike information criterion (AIC) is expressed as:

$$AIC = \frac{1}{T}(-2\ ln(\mathscr{L}(\hat{\theta}|y)) + 2K) \tag{2.11}$$

where K is the number of estimated parameters (the parameters fixed to zero are not counted), T is the total number of observation we are using to estimate the model and $\mathscr{L}(\hat{\theta}|y)$ is the likelihood of the estimated parameters given all the observations y, so, the maximum likelihood obtained for the model.
The Bayesian information criterion (BIC) is instead:

$$BIC = \frac{1}{T}(-2\ ln(\mathscr{L}(\hat{\theta}|y)) + K\ ln(T)) \tag{2.12}$$

For a deep understanding look at Burnham and Anderson (2002) [11]. It is clear that both the criteria consider the likelihood as the main measure of goodness of fit of the model, while penalizing the number of the parameters to avoid overfitting. Generally, the BIC puts much more weight on the penalization of the number of parameters, preferring more parsimonious models than the AIC. The ideal is to find the model with the lowest AIC and BIC. When this is not possible, human intervention is required. While we prefer the model with the lowest AIC, the choice should not be automatic. If the best model selected by AIC is the worst for BIC, then maybe the second best for AIC is a better choice. In general, we select models that are good for both the measures, giving more weight to the AIC.

Once we have our best candidate, it needs to pass the diagnosis phase [19]: ACF of standardized residuals and squared standardized residuals, chi-squared goodness of fit of standardized residuals with the theoretical distribution, ARCH-LM test for the adequacy of the GARCH fit and Ljung-Box test for the adequacy of the ARMA fit. In this last part, the analysis is left to the researcher and human interpretation. If the model does not seems to have big problems, we have finally our best choice.
The aim of this process was to select an ARMA that models the mean without too many simplifications about the variance or the residuals, estimating its parameters under the right hypothesis. Of course, it is possible that our best model, while being good on the training data, will be not suitable for forecasting. This is a real possibility given our choice to not continuously re-estimate the model as new data became known. Moreover, a financial time series is ruled by real-world events that can change the generating process of the data in different periods. It is the final aim of this thesis to see if it is possible to improve the forecasting ability of some fixed models, estimated only on a set of training data, by means of combination. It is so necessary to go beyond the ARMA models and present a class that works in a very different way.

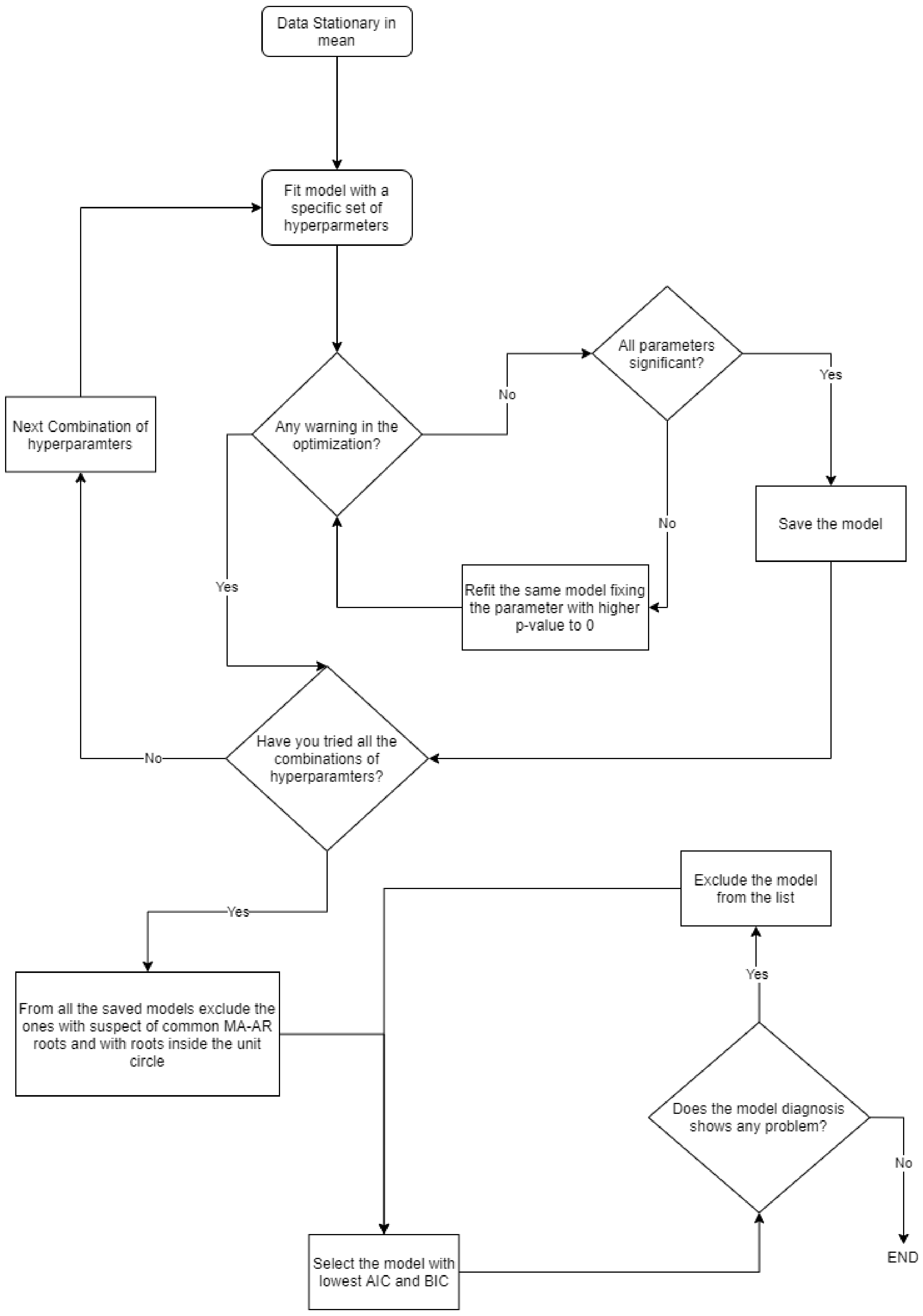

Data Stationary in mean
Fit model with a specific set of hyperparmeters
Any warning in the optimization?
No
All parameters significant?
Yes
No
Save the model
Refit the same model fixing the parameter with higher p-value to 0
Yes
Next Combination of hyperparamters
No
Have you tried all the combinations of hyperparamters?
Yes
From all the saved models exclude the ones with suspect of common MA-AR roots and with roots inside the unit circle
Exclude the model from the list
Yes
Does the model diagnosis shows any problem?
No
END
Select the model with lowest AIC and BIC

Figure 2.1: ARIMA-GARCH model selection scheme

2.2 Neural Network

2.2.1 Structure

The second class of models that we consider are the Neural Networks (NN). The structure of the Artificial Neural Network (ANN) is inspired by the biological functioning of the brain in the sense that they connect some units, called neurons, organized in different layers with some sort of synapses. Here we focus on the standard form of neural network that is the Feedforward Neural Network, specifically on densely connected Multilayer Perceptron (MLP). This means that the connections are never cyclical and all the neurons of one layer are connected to all the neurons of the next layer. In this section our main reference is the book of Bishop ”Pattern Recognition and Machine Learning” (2006) [6]. In our work the notation is slightly different due to the necessary harmonization to time series analysis. The layer zero of the network is composed of inputs and is therefore called the input layer. The last layer is the output layer and the layers between them are the hidden layers [44]. A graphical representation of a single hidden layer MLP with one output is given in figure 2.2.

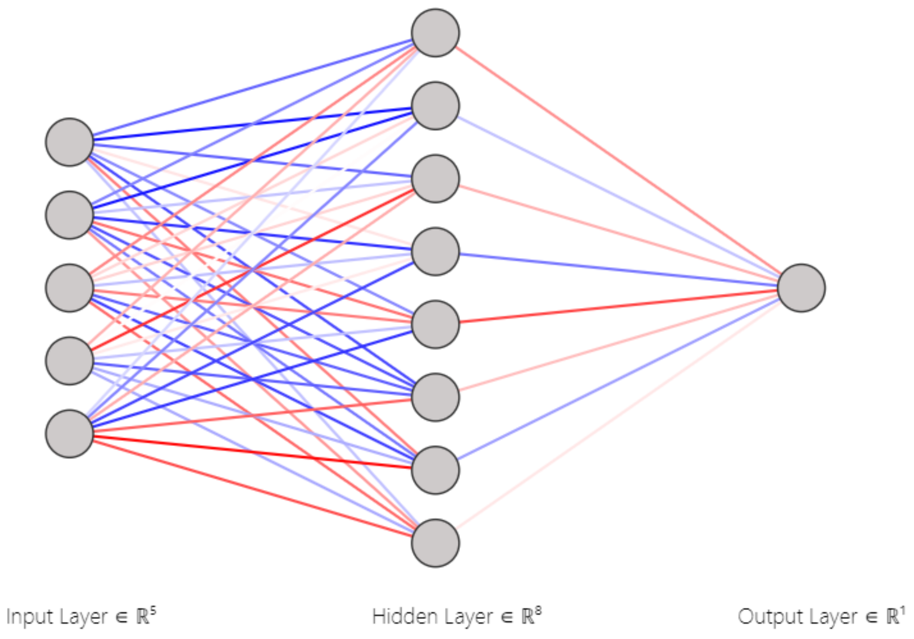


Figure 2.2: Graphical representation of a MLP with 5 inputs, 8 neurons on the hidden layer and one output.

Notice in the image 2.2 that all the elements are densely connected to the elements of the next layer. The neurons are nothing else than a function that takes as inputs the outputs of the previous layer and returns a single number. The connection between the elements represents the weight that is applied to the element from which the connection starts. Every single neuron of the first layer, namely the hidden layer because the input layer is not counted as a layer of neurons, takes all the inputs and multiply each one by a weight that is represented by a connection. In the image, a blue connection corresponds to a positive weight while a red one to a negative weight. A higher colour intensity means a higher absolute value of the weight. Each neuron, after having multiplied all the elements of the previous layer by the corresponding weights, sums the results and add a real number called "bias", obtaining a single real number. To this real number, a nonlinear function is applied to obtain the output of the neuron, which can be used as input for the next layer.

So, a single neuron linearly combines all the elements of the previous layer and then apply a nonlinear function to the result. Writing the output in functional form for the j-th neuron of the first layer (remember that the input layer is not counted), we obtain:

$$z_j^{(1)} = h(a_j^{(1)}) \tag{2.13}$$

$$a_j^{(1)} = \sum_{i=1}^{D} w_{ji}^{(1)}\, x_i + w_{j0}^{(1)} \tag{2.14}$$

where $j = 1...M$ (M neurons on the hidden layer), the superscript "(1)" means that we are on the first layer and the number of inputs x_i is D. $w_{ji}^{(1)}$ is the specific weight that connects the j-th neuron of the first layer with the i-th input, while $w_{j0}^{(1)}$ is the bias of the j-th neuron. $a_j^{(1)}$ is the activation of the neuron and $h()$ is a nonlinear differentiable function that, when applied to the activation, returns the output $z_j^{(1)}$. Each neuron in the next layer does the same thing, but it receives as inputs the outputs of the first layer. A different activation function can be chosen for each layer.

It should be clear that the number of layers can be increased indefinitely following the same logic: every neuron of each layer is a nonlinear function of all the elements of the previous layer that are themselves nonlinear functions of all the elements of the layer before etc... . This greatly increases the complexity of the network and brings us into the world of deep learning [32]. Fortunately, a great number of layers is not strictly necessary thanks to the universal approximation theorem showed by Hornik (1991) [27] that tells us that a multilayer feedforward network with only one hidden layer is a universal approximator of any continuous mappings over a compact input domain, given a sufficient number of hidden units. The result was further generalized by Leshno et al. (1993) [33] showing that this property holds for all non-polynomial activation functions. The reason for using a deep neural network is that it needs a much smaller number of neurons than networks with one hidden layer, when learning complex tasks [34]. In our case, we are interested in the forecast of a time series and in our many trials on our data we have not been able to find any justification for using a neural network with more than a single layer. Expanding the number of units of a single layer was enough to reach the same result of a two or three hidden layers network. Citing the ”introduction to neural network” of Svozil et al. (1997) [44]: ”there is no theoretical reason to use more than two hidden layers.[...]It is strongly recommended to use one hidden layer and then, if using a large number of hidden neurons does not solve the problem, it may be worth trying the second hidden layer”. Given our experiments, this last recommendation and the universal approximation theorem, in this work we use only neural networks with a single hidden layer, expanding its number of neurons.

Once defined the number of layers we have to define what are the inputs and the outputs. The structure of the neural network presented can be applied to time series forecasting giving as inputs a certain number of past values of the series and as outputs a future value [29]. For a review of the recent literature about the application of neural networks to financial time series, look Sezer et al. (2020) [41]. In this work we are interested only in one-step forecast so it is natural to have only one output unit in the output layer and D previous values of the series as inputs. In the output we choose the identity function $f(x) = x$ as activation function because we need the output of the network to be unbounded. Substituting the lagged series to the input our network function becomes:

$$\hat{y}_t = \sum_{j=1}^{M} w_j^{(2)} h(\sum_{i=1}^{D} w_{ji}^{(1)} y_{t-i} + w_{j0}^{(1)}) + w_0^{(2)} \tag{2.15}$$

We remember that M is the number of hidden units in the single hidden layer, D is the number of inputs (the number of lags we use to model the variable y), the activation function of the output is the identity function and $h()$ is a nonlinear differentiable function.

In the image 2.3 there is a representation of some weights and neurons.

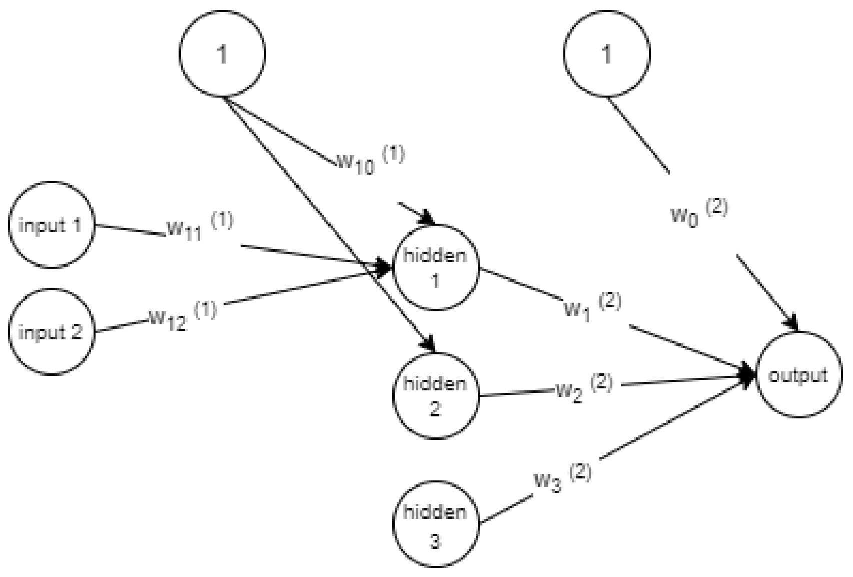


Figure 2.3: A visualization of some weights and some neurons in a simple neural network

This neural network that uses the lagged series as inputs and returns the estimation for the next time step is a nonlinear autoregressive model. The next step is to find the best parameters of this model.

2.2.2 Fitting the network

Fitting the network means finding the combination of weights and biases that maximize the likelihood function of the data. We can give a probabilistic interpretation of a neural network as a model that models the mean of a distribution conditional on some inputs. Defining y_t the target variable, X_t the input (that is a $D \times 1$ vector of the values of the series before time t), $\hat{y}_t$ the output of the neural network, W the weights (the biases are included in the weights) and σ^2 the variance of the target variable we can write [6]:

$$p(y_t|X_n, W, \sigma^2) = \mathcal{N}(\hat{y}_t(X_t, W), \sigma^2) \tag{2.16}$$

It is made explicit as $\hat{y}$ is a function of the inputs and the weights. The log-likelihood given T observations would then be:

$$-\frac{N}{2}ln(2\pi) - \frac{N}{2}ln\sigma^2 - \frac{1}{2\sigma^2}\sum_{t=1}^{T}[\hat{y}_t(X_t, W) - y_t]^2 \tag{2.17}$$

In a Gaussian framework, maximizing the likelihood corresponds to find W that minimize the sum of squared errors. Once we have the estimates of the weights we can find the variance σ^2 by maximizing the likelihood. This is a two-step procedure. Of course, given the complex nonlinearity of the network function, the optimization process of the target function cannot be solved analytically so an iterative process is needed. Further complexity is that multiple combinations of weights can produce the exact same result, so there is not a single optimal solution [6].
The standard iterative process used is the gradient descent that at every step change the weights in the direction of the negative gradient going towards a local minimum [6] [38]. The updated weights are:

$$W^{\tau+1} = W^{\tau} - \eta \nabla_{W^{\tau}} L(W^{\tau}) \tag{2.18}$$

Where L is the loss function, defined as the sum of squared errors, $\nabla_{W^{\tau}}$ is the gradient with respect to the weights computed at the previous step and η is the learning rate that controls the width of the change in weights at each step τ. Here we choose to use the batch gradient descent approach that performs a single update per epoch computing the gradient on the whole data set. The number of epochs is the number of times that our model uses the entire data set to update its parameters. With the batch gradient descent, only one update is performed for each epoch. This approach is generally more precise but slower and unfeasible if the whole data set is larger than our machine memory [38]. Moreover, we do not use the basic gradient descent presented in the previous equation that has a single fixed learning rate that does not change between parameters or in time. Instead, we use a more recent method called Adaptive Moment Estimation (ADAM) that uses a different adaptive learning rate for each parameter. This should converge faster to a local minimum. We refer to the original work Kingma and Ba (2015) for details [28].

Once the optimization algorithm is defined, we need to compute the gradient of the loss function to perform it. The standard algorithm for this purpose is called error back-propagation [44] [6] [29]. While the term "forward propagation" refers to the evaluation of the output of the network starting from the inputs, the "backpropagation" refers to the errors of the output that are propagated back through all neurons, starting from the last unit. Suppose that we want to update the weights using a single observation y_t (online learning). First, the network function should be evaluated using forward propagation of the input data that flow through the network computing the output of all neurons on the hidden and output layer. We remind that we call $a_j^{(l)}$ the linear combination of inputs that the neuron j in the l-th layer receives, while we call $z_j^{(l)}$ the respective nonlinear transformation of $a_j^{(l)}$ provided by the activation function $h()$. To compute the derivative of the loss function of this single sample with respect to a specific weight $w_{ji}^{(1)}$ of the hidden layer, we should apply the chain rule:

$$\frac{\partial L_t}{\partial w_{ji}^{(1)}} = \frac{\partial L_t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial z_j^{(1)}} \frac{\partial z_j^{(1)}}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial w_{ji}^{(1)}} \tag{2.19}$$

Where L_t is the loss that the network experienced for the observation at time t. In our case it is a quadratic loss $L_t = \frac{1}{2}(\hat{y}_t - y_t)^2$, given a single neuron on the output layers. In equation 2.19 we did an example of chain rule for a weight of the hidden layer. Computing the derivatives we obtain:

$$\frac{\partial L_t}{\partial w_{ji}^{(1)}} = (\hat{y}_t - y_t)\, w_j^{(2)}\, h'(a_j^{(1)})\, y_{t-i} \tag{2.20}$$

This is only for a single observation at time t. If we want to compute the derivative of the weight for all the T training samples, we have to sum them:

$$\frac{\partial L}{\partial w_{ji}^{(1)}} = \sum_{t \in T} \frac{\partial L_t}{\partial w_{ji}^{(1)}} \tag{2.21}$$

And this is only the derivative of a specific weight. Applying the same process to all weights and biases, we can compute the entire gradient used by the gradient descent algorithm. For details refer to Bishop (2006) [6] where the computations are slightly different from ours but the results are the same.

2.2.3 Selection

After having had a taste of how the estimation of the parameters is done, it is necessary to choose the best neural network. The possibilities in the choice of the hyperparameters are incredibly large. We can choose the number of layers, the number of neurons in each layer, the activation function for each layer, the number of inputs, the number of outputs, the regularization process and the optimization process. Of course, it is not feasible to test for all the combinations, so the majority of choices are fixed a priori with a justification based on experience and on existing literature. We want to underline that these choices are not done with the specific aim of modelling the VIX data. The choices should be general enough to be maintained with a large number of time series.

We have already discussed the limitations that we have decided to put on the number of layers. These are based on the universal approximation [27] [33] theorem and on the suggestions of Svozil et al. (1997) [44]. We will use only networks with a single hidden layer and an output layer.

The choice of the optimizer too will be kept fixed to ADAM that should be flexible enough for every network that we build [28].

The number of inputs will be fixed to the maximum order of AR that we try during the ARIMA-GARCH selection process. This does not mean that the AR order of the selected ARIMA-GARCH will be the same, but at least both the models have the opportunity to go back in time by the same amount. In the neural network, we keep the number of inputs fixed instead of trying a different number of inputs as we do with ARIMA-GARCH. This is because the network can easily set the weight to zero for all the information that are not useful, mainly thanks to the activation function that we use.

The number of output is fixed to one, exactly as in ARIMA-GARCH. We are interested in the forecast of the next time step. Moreover, given the fact that we want to do a regression and not a classification, the activation function of the output unit will be the identity function.

The choice of the activation function used in the hidden layer is particularly complex given the vast amount of possibilities. In this work we will use only the Rectified Linear Unit (ReLU) activation function defined as $h(x) = max(0,x)$ that is widely used, has many useful mathematical properties and is biologically justified [20]. This function is nonlinear in the entire domain but linear for positive numbers. This property makes the computation of its derivative extremely easy, with a 0 derivative on all negative numbers and equal to 1 for positive numbers. It is non-differentiable on 0, but this limitation can be overcome by setting the derivative on 0 equal to 0 or 1, so using the derivative of the left or right point next to 0. On different tests that we have performed on our specific set of data, the ReLU performed better than other alternatives such as the logistic sigmoid function on both the generalization capability and the speed of optimization. The most interesting characteristic of the ReLU is that it sets to zero all the units with a negative activation while being linear for the others. This means that for a particular combination of inputs, the network becomes a linear function with some neurons active while, for another combination, the active neurons can be different ones. So the network becomes a sparse network that is composed of many ”switches” that activate only for some inputs. For a complete analysis of the advantage of this sparsity, refer to the fundamental work of Glorot et al. (2011) [20]. In a very recent preprint by Szandała (2020) [45] all the most common activation functions have been compared, finding the ReLu to be the first choice overall.

It should be clear now that a neural network large enough can fit extremely well the training data, thanks to its flexibility and its great number of parameters. The problem is the risk of overfitting the training data, modelling also the randomness and losing generality. Here we are mainly interested in the forecasting ability of the neural network on new data so it is particularly important to not overfit the training data. To show how easy is to overfit the data with a neural network, we present an example in the next figure 2.4. This is a very small portion of the data that we will use in Chapter 3. The neural network of this example is built following everything we have said since this point. The inputs are the 5 previous values of the series and the number of hidden units is 200. The network has been trained through 5000 epochs, so it has seen the whole training set 5000 times, updating the weights 5000 times towards a minimum of the loss function. The period presented in the image is a period of high volatility in the series and it is clear that the neural network is modelling also the randomness of the data, making it difficult to distinguish the fitted values from the real values. Of course, this network has poor performance on data that are not in the training set.

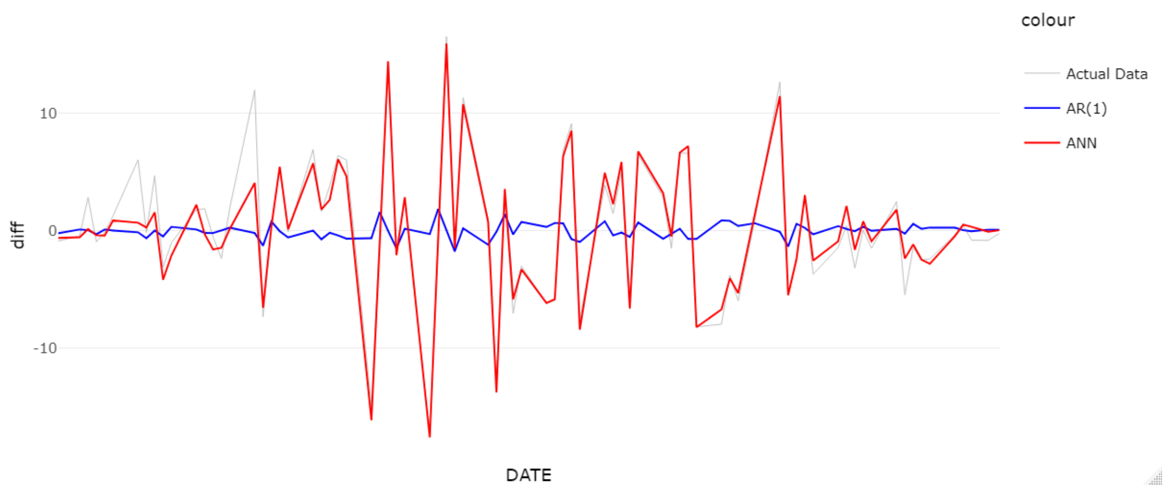


Figure 2.4: Example of overfitting in training data. The actual time series is the grey line, the values fitted by the NN are in red and the blue line are the values fitted by an AR(1)

To avoid this clear risk of overfitting, we use two compatible, very simple but powerful regularization methods. The first is called dropout and consists of randomly considering some neurons and all their connections equal to 0 at each step of the training with a fixed probability. This means that at each step every neuron has a probability of being dropped resulting in a reduced neural network with only some active neurons. At the next step, we randomly set to zero another set of neurons and we train a new network, that it is different from the one trained at the last step. Of course, the dropped neurons are considered equal to zero only in that specific step. Then their previous values are restored. At the end of the training time, we use all the neurons together instead of dropping a part of them as in the training phase, so we have to rescale the weights, that are too big, multiplying them by the inverse of the probability of dropping each neuron. This method trains a bunch of different neural networks that share some parameters and then average their results when doing prediction outside the testing data. This is equivalent to a sort of model combination and is useful to avoid overfitting the data and losing generality. For details refer to the original works [43] [26]. Both the works suggest using a 50% drop rate in the hidden units and 20% on the input layer. Here we are modelling time series so every input is extremely important. So we only randomly drop the hidden units assigning to each hidden unit a 50% probability of being dropped at each epoch.

The second regularization strategy that we use is called early stopping [6]. The aim of the training is to minimize the loss function on the training data, but the flexibility of the neural network permits to model also the randomness of these data, resulting in a very high loss on data for which it is not trained. This can be prevented in a very simple way by randomly select some training data as validation data and check during training how the model performs on these data that it cannot uses to update its parameters. This permits checking the ability to generalize on data not visible at the moment of training. The number of epochs describes how many times the model go through the entire training set to adjust its weights. Of course more the epochs, more the possibilities to minimize the loss function. If we let the network learn from data too many times the risk of overfitting them is high. This is why we use the early stopping strategy and, instead of letting the model reaching an extremely low loss on all the training data, we stop the training when the loss on validation data stop decreasing. We suppose that to be the optimal point in terms of generalization.

The last hyperparameter left to choose is the number of neurons in the hidden layer. For this there are no fixed rules so we try all the possibilities between a minimum and a maximum number, choosing the model with the lowest mean squared error on the validation data (that cannot be used for training). We fix the minimum number of hidden units equal to the number of inputs while the maximum number is chosen with a rule of thumb that requires 10 observations for each parameter. With D inputs, M hidden units and 1 output, the total number of parameters of the network is $D \times M + M + M + 1$. Given T training data and requiring 10 observation per parameter, the maximum number of hidden units is $M = (\frac{T}{10} - 1)\frac{1}{D+2}$.

The model selected as being the best on validation data is then retrained some other times to check the stability of the network. This last phase is left to the researcher sensibility. If the network passes this last step, we finally have our best neural network.

2.3 Hybrid Model

The last class of model that we want to treat is a form of hybrid models that uses both the ARIMA-GARCH and the Neural Network model. As previously seen, the two models are very different in their structure and in their estimation process and are good at different things. With their work on time series forecasting with Neural Networks, Tang and Fishwick (1993) [46] found the feedforwards nets to perform better than the Box-

Jenkins model on series more irregular and with short-memory. The Box-Jenkins model instead was slightly better on series with long-term memory. This is only an example of a possible difference, but it is possible that many others exist.

The idea of merging these two types of models is based on the fact that, while being very good and robust in forecasting linear time series, the ARIMA model is not able to extract nonlinear dependencies. For this reason, it has already been proposed the use of a neural network to model the residuals of the ARIMA model [49] [47]. In particular, the idea for the model used in this thesis is based on the intuition of Zhang (2003) [49] of considering a time series as a sum of a linear and a nonlinear components.

$$y_t = Linear_t + Nonlinear_t \tag{2.22}$$

Where $Linear_t$ is the linear series at time t and $Nonlinear_t$ is the nonlinear part at the same time. If we fit an ARIMA that is able to model the linear component, then the residuals can be used as inputs for the neural network that can model the nonlinear part. In this thesis the ARIMA model is estimated considering also the heteroscedasticity, using an ARIMA-GARCH approach and trying to found the model that is most correct from a statistical point of view. We follow the process presented in the section about ARIMA-GARCH. Once we have this model, we model the residuals of the ARIMA-GARCH as a separate time series with a neural network that uses the last D residuals to predict the next one. Then our prediction for the original time series is the sum of the predictions of the ARIMA-GARCH and of the neural network. The number of inputs of the network is equal to the maximum order of the AR that is tried for the ARIMA-GARCH model during the selection. The characteristics and the selection of the neural network are the same presented in the section about neural networks.

This hybrid model is composed of two steps both for estimation and forecast. This can lead to a suboptimal parameter combination when compared to a joint estimation of the two models. Unfortunately, the complexity of the neural network, combined with the necessary regularization used for it, makes the one-step estimation difficult.

The complete model can be written as:

$$y_t = c + \sum_{i=1}^{p} \phi_i \, y_{t-i} + \sum_{j=1}^{q} \theta_j \, \varepsilon_{t-j} + \varepsilon_t \tag{2.23}$$

$$\varepsilon_t = \sum_{j=1}^{M} w_j^{(2)} h(\sum_{i=1}^{D} w_{ji}^{(1)} \varepsilon_{t-i} + w_{j0}^{(1)}) + w_0^{(2)} + \eta_t \tag{2.24}$$

Where η_t is a random error term. We underline again that in the two step approach, the parameters of the ARIMA-GARCH are estimated ignoring the neural network and considering the variance of ε_t time varying thanks to the GARCH. In the second step we consider the ε_t as being generated by a distribution with fixed variance and the mean determined by the output of the neural network.

Chapter 3

Model Combination

3.1 Introduction

The idea of combining forecasts from different models to obtain a better result is today a well-established concept. The paper of Bates and Granger in 1969 [5] explored for the first time this possibility by discussing the decision about the optimal weights to assign to each forecast in a linear combination of them. For a more recent discussion look [22]. They wrote that the idea of discarding all the model except the best one is not optimal because all the models can contain independent information. In their words the two types of information are:

- ”One forecast is based on variables or information that other forecast has not considered”
- ”The forecast makes a different assumption about the form of the relationship between the variables”

In the case treated by this thesis, the first point is not of main interest because it was our cure to give to the models the same access to information about the past. The second point is instead really interesting due to the fact that the models we have presented are very different in nature. The ARIMA-GARCH model was estimated considering also the heteroscedasticity and different possible distributions of the residuals, trying to build a model theoretically strong. The Neural Network is instead much more a ”black box” that trade the loss of interpretability with high flexibility to model also non-linear data. During the estimation of the neural network the residuals are treated as homoscedastic and normally distributed. Besides the different hypothesis, also the optimization is done in a very different way, with the neural network using the backpropagation algorithm. Moreover, we introduced also a hybrid model that stacks together the ARIMA and the Neural Network approaches considering the time series as composed by a sum of a linear and a non-linear part. Also this model behaves differently from both the previous models. Given the notable differences in the structure of our three models, and on their assumptions, the second type of independent information [5] should be present, opening the possibility to further improve every single model with a combination of them.

An important point to consider is that all these model have pretty high levels of complexity when compared to the most basic models. While this is very useful to fit the training data, it can be less effective while forecasting periods that are very different from the training data. In such cases it is possible to have a very simple model performing better than the complex ones. The article of Green and Armstrong (2015) [24] makes a general analysis of complexity vs simplicity based on the findings of many other papers. They found the simplest methods to be hardly beatable by more complex ones on forecasting ability. For this reason, we introduced among the used models also the simple AR(1) (with constant) fitted on the same training data of the other models.

We end up with four different models: a neural network, an ARIMA-GARCH, a hybrid model and an AR(1). The hybrid model is a particular one because it is based on the ARIMA-GARCH and then uses a neural network to model the residuals. If the neural network is built in the right way, this method should always be better than the original ARIMA-GARCH, mainly for its ability to extract also nonlinear pattern that the ARIMA-GARCH cannot handle. Following this logic, only three of the four models will be used in the combination: the hybrid, the neural network and the AR(1).

After having defined the models to be used for the combination, we have to find the best ways to actually combine them. There are several possibilities and, while the most complex methods can be attractive, from the review of Meneze et al. (2000) [35] we know that there are several cases where the simple average of the forecasts can be the best choice over more complex methods. The same conclusion was found by Clemen (1989) [13] that in his review found many studies where the simple average performed best or almost best. It was also in the intentions of Granger and Bates (1969) [5] to propose a simple and attractive method to make the combination appealing. For these reasons we include in this work some well-established and relatively simple methods to combine forecasts, starting from the simple average and explaining why there is the need to try also optimal weights. The optimal weights can be found through a linear regression [21]. From this point, we explore also the possibility to combine the forecasts through a feedforward neural network.

3.2 Classic methods

3.2.1 Simple average

The first and most basic way to combine a number of forecasts is the simple average of the forecasts made for a specific time. If we call y_{t+1} the true value of the series at time $t+1$ and $\hat{y}^{k}_{t+1|t}$ the forecast made by the $k-th$ model for $t+1$ using all the information until t, then our combined forecast $\hat{y}^{c}_{t+1}$ is:

$$\hat{y}^{c}_{t+1} = \frac{1}{k}\sum_{j=1}^{k} \hat{y}^{k}_{t+1|t} \tag{3.1}$$

With this method, we will give equal weight to all the forecasts without adding any constant or external term. This means that if the individual estimators are biased, also the combined one will be biased (if the biases do not compensate for each other). This can be acceptable for us, given that we are mainly interested in the practical utility of this combination method. In real-world data, more than unbiased estimators, we privilege estimators that are able to consistently provide low errors. The main metric to evaluate the error that we are using throughout this thesis is the mean squared error (MSE), supposing that in a real-world application it is desirable a model that reduces the errors penalizing the large ones, which are dangerous in every industrial or financial plan. If we consider the real data to be generated by our estimate plus an error term ε with variance σ^2, we can rewrite the expected squared error decomposing the variance and the bias:

$$E[(y-\hat{y^c})^2] = \sigma^2 + E[V[\hat{y^c}]] + E[(y-E[\hat{y^c}])] \tag{3.2}$$

The complete derivation can be found in Shakhnarovich notes (2011) [42]. So the expected squared errors can be seen as a sum of the variance of the error term (the randomness that cannot be deleted by the model), the expected variance of the estimation and the expected bias of the estimation. This result in the well-known trade-off between bias and variance: where the optimal is the unbiased estimator with the least variance, in a real application we can accept a biased estimation if this leads to a diminished mean squared error thanks to the decrease of variance.

To see if it is possible to shrink the variance with a simple average, think of a combination of only two unbiased forecasts series. The variance of the errors of the combination would be (we drop the time subscript):

$$V[y-\hat{y}^c] = \frac{1}{4}(V[y-\hat{y}^1] + V[y-\hat{y}^2] + 2COV[y-\hat{y}^1, y-\hat{y}^2]) \tag{3.3}$$

Rewriting using the correlation ρ:

$$V[y-\hat{y}^c] = \frac{1}{4}(V[y-\hat{y}^1] + V[y-\hat{y}^2]) + \frac{1}{2}\rho\sqrt{V[y-\hat{y}^1]V[y-\hat{y}^2]} \tag{3.4}$$

If we fix the two variances, we can see that this becomes a linear function of the correlation, where the minimum variance of the combination errors is reached when the correlation is -1, while the maximum is reached when the correlation is $+1$. If now we fix the correlation and we consider the variance of the combination errors as a function of the two individual variances that we call σ_1^2 and σ_2^2, we can analyze what would happen in the best and worst-case scenario. In the best case, when the correlation is -1, the error variance of the simple combination is less than both the individual variances if $\frac{1}{9}\sigma_1^2 < \sigma_2^2 < 9\sigma_1^2$. This solution can be found by solving the system with $\rho = -1$:

$$\begin{cases} \frac{1}{4}\sigma_1^2 + \frac{1}{4}\sigma_2^2 + \frac{1}{2}\rho\sigma_1\sigma_1 < \sigma_1^2 \\ \frac{1}{4}\sigma_1^2 + \frac{1}{4}\sigma_2^2 + \frac{1}{2}\rho\sigma_1\sigma_1 < \sigma_2^2 \end{cases} \tag{3.5}$$

As an example imagine having two individual variances equal to 1 and 9. In this case, the error variance of the combination will be exactly equal to the lowest individual variance. If the variances are 1 and 10, then the simple combination is not able to reduce the variance below the one of the best model.

In the worst-case scenario, with a correlation equal to 1, the error variance of the combined estimator is never lower than the best individual variance. To check this result, solve the system with $\rho = 1$.

This was to show that the simple average combination, while suffering from biased individual forecasters, it also does not assure always the reduction of variance and of the mean squared error. In the case of a high positive correlation, this kind of combination is worse than the best of the two single models. Also in an ideal scenario with a large negative correlation, is not always true that the simple average will be better than the best single forecaster. We said that we can accept a biased combination in exchange for a diminished variance and this is the case only in some specific situations. Anyway, our reasoning is limited to the in-sample variance. In many cases, the simple average has shown to beat other methods on the out-sample performance [35] [13] [24]. See the work of Blanc and Setzer (2016) [7] for a complete analysis about when to choose the simple average. In general, while not optimal in sample, this method does not need the estimation of any parameter, so it does not need a minimum amount of data and it is stable in time and robust also in situations where there is a change of the state of the world (e.g. periods of high volatility).

3.2.2 Constrained regression

The second method that we use is a consequence of one of the problems of the simple average. Instead of using a simple average that diminishes the variance of the combination only in some cases, we lose some simplicity and introduce a vector of weights chosen to minimize the variance of the combination errors. If our series is $y' = (y_1, ..., y_T)$ and we have K competing forecaster, then our forecast matrix is a $T \times K$ where each column is the forecasted series for each model. We call it F. Then we want to minimize the variance of the errors of the combined model using a weight vector w that is $k \times 1$. As with the simple average, we require the weights to sum to 1. Then:

$$V[\varepsilon_c] = E[\varepsilon_c' \varepsilon_c] = E[(y - Fw)'(y - Fw)] \tag{3.6}$$

Of course we don't know the real variance, so we estimate it on our data. So to minimize the variance we have to minimize the sum of squared errors of the combined forecasts, with a constraint on the weights which should sum to one. Granger and Ramanathan [21] shows that the estimation of these weights can be performed with an ordinary least squares regression without constant that regresses $(y - f_k)$ on all the $(f_j - f_k)$ with $j \neq k$ where f_j is the series of forecasts made by the $j-th$ model (the $j-th$ column of F). The OLS regression assures us that the weights are chosen to minimize the sum of squared errors, and this corresponds to minimize the variance of the errors. Moreover, the constraint that requires the sum of weights to be equal to 1 is simply done by assigning to the forecasts f_k a weight equal to $1 - \sum_{j \neq k} w_j$. Of course the regression for the optimal weights is not possible prior to the observation of all the real data we want to predict. We follow two approaches to overcome this problem. With the first we find the optimal weights on the training set, performing the regression on the fitted values and doing the strong hypothesis that the behaviour of the forecasts errors will be the same of the residuals, so we fix the computed weights to use throughout the test data. With the second approach instead, we relax the constraint of constant weights and we perform a rolling regression on a moving window of length m of test data. Since we use the previous m forecasts and actual data to estimate the optimal weight for the next period, then for the first m forecasts of the test series we will use some data and fitted values that are in the training set. The length of the window is hard to fix a priori without looking at the testing data. In general, given the high number of testing data and the variability of data in different periods, we will prefer stability of the weights so the window should be long.

3.2.3 Unrestricted regression

As it is clear from the computation of the variance 3.6, we are assuming all the forecasters to be unbiased. Of course, this is not always the case. As we said before, obtaining a biased combination of forecasts that is able to reduce the squared errors is not a big deal for us who are interested in practical applications. It is anyway the case to add in the poll of our candidates combination methods also an unbiased one. The idea comes again from the work of Granger and Ramanathan (1989) [21], where they proposed the use of a constant term in the regression to correct the possible bias present in the individual forecasts.

In this case, the sum of the weights should not be constrained to sum to one. This time again, minimizing the variance of the errors of the combination is obtained by minimizing the sum of squared errors. Defining as w_0 the constant term and $\mathbf{1}$ as a vector of ones of appropriate dimension, the optimization problem becomes:

$$min\ (y - w_0\mathbf{1} - Fw)'(y - w_0\mathbf{1} - Fw) \tag{3.7}$$

We can simplify the notation by adding a column of 1 as first column of the matrix F and adding the constant w_0 as first element of the weight vector w. So we can rewrite the optimization problem:

$$min\ (y - Fw)'(y - Fw) \tag{3.8}$$

The solution is straightforward and we obtain the OLS estimator $w = (F'F)^{-1}F'y$. In the paper of Granger and Ramanathan (1989) [21] is it shown that this regression gives smaller errors of the regression without constant (with or without weight constraints) and it is unbiased. In this way we have relaxed the request of unbiased individual estimators. As previously done, we apply this methodology both in a static and dynamic way. In the first case, we make the strong assumption that the behaviour of the fitted residuals on the training set is the same of the forecast errors, so we perform the regression on the training data and keep the weights fixed. With the second approach, we do a rolling regression in a window of fixed length. This length is again chosen intuitively privileging stability of weights over fast changes. For the first forecasts, where we do not have enough test samples for the regression, we will use the last training samples.

We end up with 5 different methods whose performance will be compared on test data: the simple average (SA), the regression with weight constraint and without constant both with static and dynamic weights (respectively SREG and DREG) and the unconstrained regression with constant in both its static and dynamic version (SUREG and DUREG).

3.3 Combination Using Neural Network

Since this point, we have considered only the possibility of a linear combination of the forecast. Now we can extend our horizon further by using a general nonlinear approximator as a neural network of the same type of the one described in Chapter 1. The use of neural networks to combine forecasts of other models has already been explored. Kamstra and Donaldson (1996) [17] found their combination done with Neural networks to beat the classical linear models in terms of errors. The same results have been shown in Harrald and Kamstra (1997) [25] and, more recently, in Aladag et al. (2010) [4].

The last method that we proposed in the previous section was based on a linear regression of the real values on the different forecasts. This OLS regression can be viewed as a neural network that uses only linear activation function. Starting from this concept of regression, we can generalize it by using a nonlinear function as activation in the hidden layer and performing so, a sort of nonlinear regression. So we follow the logic proposed by Granger and Ramanathan (1984) [21] but using a nonlinear regression that, again, minimize the mean squared error.

The neural network for the combination follows the same rules presented in Chapter 1 for the other neural networks used in this thesis. The difference is that now we use as inputs the forecasts of the three models we are combining. At time t the network receives as inputs the three forecasts for that time and tries to combine them to better approximate the real value y_t. This is the same done by Aladag et al. [4]. The number of hidden neurons will be fixed to six with a dropout rate of 50%. This means that at each step in the training of the network, we will train a neural network with 3 active neurons. The network will be trained on the training set a single time. Then the same network, with fixed weights and biases, is used in the test set to combine the forecasts of the three models.

Chapter 4

Empirical Results

4.1 VIX and data description

4.1.1 Introduction

The VIX (CBOE Volatility Index) is an index provided by the Chicago Board Options Exchange about the expected volatility of the S&P 500 for the next 30 days. Based on the idea of a volatility index proposed by Brenner and Galai (1989) [10], the VIX estimates the 30 days expected volatility using the implied volatility derived from the market price of put and call options on the S&P 500, with more than 26 days and less than 37 days to expiration. The price of the options represents the market expectation about future volatility. The details and the computation of the index can be found on the CBOE white paper about the VIX [12].

The interest in this index is given by its ability to explain how the market ”feels” about the near future. For this reason, it is also called the ”fear index”. Ahoniemi (2006) [3] used ARIMA and ARIMA-GARCH models augmented with external regressors to predict the next day direction of the VIX index. Using no external regressor he reached a 55.3% precision maximum. In another work on the one-day forecast of the VIX index, Stavros (2008) [15] showed as using the realized and the conditional volatility was not useful to improve the VIX forecast, concluding that the VIX is hard to forecast. Konstantinidi et al. (2008) [30] used different models to forecast the day on day difference of different implied volatility indexes (VIX included). They found the models to be pretty similar on the out-sample forecasting ability of the VIX index, at least following the MSE metrics. The directional accuracy (share of correct prediction of the sign of the first difference of the series) for the VIX has been found to be the highest (54.71%) using external economic regressors, while, using only the past values of the series, the ARFIMA performed the best (53.41%). More recently Psaradellis and Sermpini (2016) [40] applied a heterogeneous autoregressive process (HAR) combined with a genetic algorithm–support vector regression(GASV) and other hybrid models. Osterrieder et al. (2020) [39] used an LSTM neural network to predict the intraday value of the VIX using only a fraction of the options used to compute the original index.

4.1.2 Data

In this thesis, we use the daily data of the VIX index from January 2, 1990 to June 9, 2021. From 1992 the VIX is computed intraday, so we use only the closing values. The data are split into a training set for the selection and estimation of models and a test set for the out-sample validation. The last day in the training set is February 25, 2015. From that point to the end of the series, all data are used only to test the models. The training set is composed of 6333 observations and the test by 1583. The series is pictured in the plot 4.1.

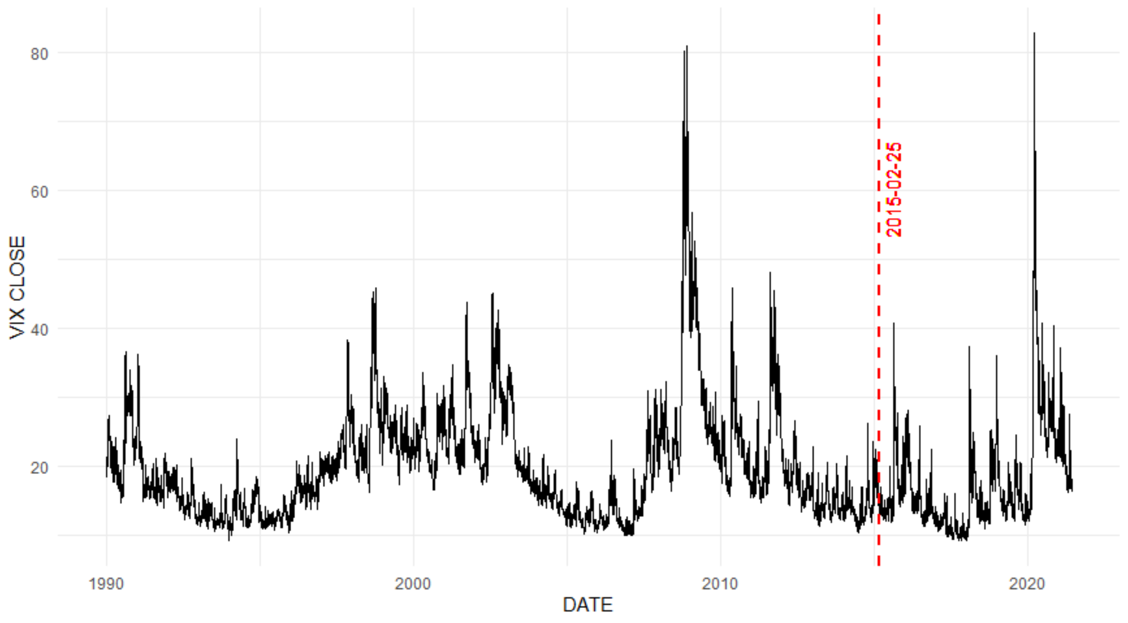


Figure 4.1: The complete time series used in this thesis, from January 2, 1990 to June 9, 2021. The red dotted line is the last day used for training

The huge spikes of the ”fear index” are clearly visible for the 2008 financial crisis and the 2020 SARS-CoV-2 pandemic. It is worth noting that one of these two extreme periods is in the training set and the other in the test set. This can create problems in the training process of models that reduce the mean squared error of the sample data uniformly because, in these extreme events, the MSE is so high that the model will give too much weight on these short periods of time, ignoring the rest. It is worth to try diminishing the magnitude of these events. For this reason, all the empirical analysis will be done in parallel for two different transformations of the VIX index. The index cannot be modelled directly due to its clear non-stationarity, so these two transformations are used to make it stationary.

The first way to do this is to compute the day on day difference of the series as done in [30]. While generally making the data stationary, this does not solve the problem of the extreme values. The second way is to compute the logarithmic return of the index to make the data stationary while diminishing the magnitude of big spikes and helping the models with data that are more similar to a normal distribution, with less extreme events characteristics of fat tails distributions. We will use both the series to compare the results between them. Calling y_t the value of the VIX CLOSE value at time t, we define:

$$\Delta_t = y_t - y_{t-1} \tag{4.1}$$

$$r_t = log_e(\frac{y_t}{y_{t-1}}) \tag{4.2}$$

Histograms of the Δ_t and r_t are provided in the plot 4.2 and some general characteristics of the series are provided in table 4.1.

	mean	median	sd	min	max	skewness	excess kurtosis
Δ_t	0.000	-0.060	1.648	-17.64	24.86	1.493	31.12
r_t	0.000	-0.004	0.067	-0.351	0.769	0.967	9.57

Table 4.1: Selected statistics of the two series Δ_t and r_t

Both the series are centered around zero and have the median that is less than the mean. This means that there are more extreme values on the positive side. This is confirmed by the skewness that is positive and generally implies a longer right tail. So the market expectations about future volatility are more prone to fast increases rather than fast decreases. The kurtosis is higher than the normal one as in the vast majority of financial time series that are susceptible to crisis or non-predictable events in general. The excess kurtosis is much higher in the series of first difference denoting fatter tails and a greater deviation from the normal distribution. As expected the logarithmic return makes the distribution more similar to a normal one. The Augmented Dickey–Fuller test confirmed the stationarity for both the series (in both cases $p-value < 0.01$, so we reject the null hypothesis of the presence of unit root). The Jarque–Bera test has been performed and in both cases the normality of the data is rejected with an extremely low p-value.

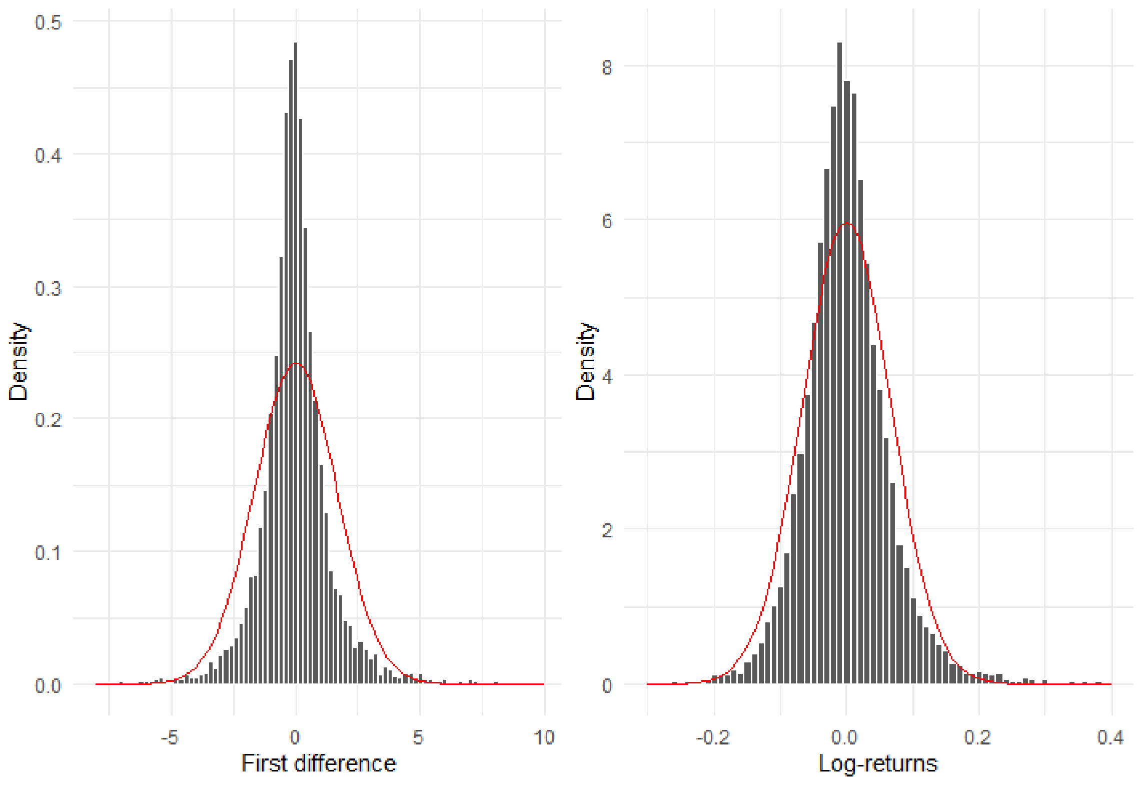


Figure 4.2: The two unconditional distributions compared with a normal (red line) with the same mean and standard deviation of the data. Δ_t on the right and r_t on the left

4.2 Model Selection and In-Sample Performance

We select the best ARIMA-GARCH model on the training data following the process described in the ARIMA-GARCH chapter and summarised in figure 2.1. The maximum order of the ARMA that we try is (5,5). All the combinations until this order are tried starting from (1,0). Our choice is justified by the fact that, during a normal week, the VIX index is computed 5 days per week (from Monday to Friday). So we expect the value at time t to be influenced mainly by the last 5 days, so from the same week-day on the previous week until the day before the one we want to predict. Moreover, due to computational limitations, it is hard for us to increase the number of trials. The best model selected for the logarithmic returns r_t is an ARMA(1,5)-apARCH(1,1) that uses a skewed Student-t distribution as conditional distribution for the standardized residuals and has 3 parameters fixed to zero due to non-significance (the constant for the conditional mean and the parameters for the second and fourth order of MA). The best model selected for the first difference series Δ_t is an ARMA(2,1)-apARCH(1,1) with a skewed Student-t for standardized residuals and no parameters fixed to zero. For both the series, the apARCH(1,1) with the same conditional distribution has been selected. Instead, it is interesting to see that the selected ARIMA orders are completely different for the two series. Both the models were the lowest AIC models and on the top five when considering BIC. This makes us think that the choice is pretty robust. The estimated parameters with further explanations and the diagnostic of the two ARIMA-GARCH can be found in Appendix B.

Next, the two neural networks are selected, one to model directly the series and the other to predict the residuals of the ARIMA just fitted. At this point, following the suggestions found in the work of LeCun et al. (2012) [31] about the efficient backpropagation, we normalize all the training and test data by subtracting the mean of the training data and dividing by the standard deviation of the training data again. The network will be trained and tested on this normalized data. The output is then rescaled back.

Every sample that is used to train these networks is composed of one value that acts as output and the 5 preceding values as inputs. So every sample is a vector of 6 elements where the first is the output and the others the inputs. We randomly select 5% of these training data as validation data giving equal probability to every sample. The networks are trained only on the samples that are not on the validation set while the latter is used to select the best network. This results in 6016 actual samples used for training and 316 for validation purposes. To clarify what a sample is, we give a representation of 3 samples:

y_t	y_{t-1}	y_{t-2}	y_{t-3}	y_{t-4}	y_{t-5}
y_{t+1}	y_t	y_{t-1}	y_{t-2}	y_{t-3}	y_{t-4}
y_{t+2}	y_{t+1}	y_t	y_{t-1}	y_{t-2}	y_{t-3}

Table 4.2: Example of three samples. Every line is a sample. The first column is the independent variable we want to predict. The other five are the inputs.

The first column is the target of the network while the other 5 are the inputs. Once we have this matrix, we select some of the lines to validate how well the network is able to perform on new data. Given the variety of behaviours on the data (normal periods, crisis etc..) it is possible that we select as validation data only data from normal periods, and this will make us select the network architecture that performs better only on that types of data.

The number of actual training data is 6016 so, using the formula provided in the neural network chapter, the maximum number of hidden units is 85. The minimum is 5 and we try all the combinations with an increment of 5 units. Then we select the architecture with the lowest mean squared error on validation data.

The estimation is done in R using Keras.

The best neural network selected to model the r_t series has 5 inputs, 25 units in the hidden layer and one output. The best network for the hybrid model (the network trained on the residuals of the ARIMA-GARCH) for the same series r_t has 65 hidden units.

The best neural network for the Δ_t is the network with 25 hidden units, so a similar structure to the one for the r_t series. The large difference is for the network used in the hybrid model. In the case of Δ_t the best one is the network with 45 hidden units. It is interesting to notice that, with both series, the optimal number of hidden units is much higher when modelling the residuals of the ARIMA-GARCH. This can be explained by the fact that the series of residuals has already be cleaned by the linear dependencies. So the network, to obtain better results on validation data, needs a more complex structure.

Also a simple AR(1) model is fitted on all training data (validation data included), one for Δ_t and one for r_t.

The three metrics we will use in this work are the mean squared error (MSE), the mean absolute error (MAE) and directional accuracy (DA). The directional accuracy is the share of observations for which the actual sign is the same of the predicted sign. For both our series, r_t and Δ_t, a positive sign means that the VIX has grown while a negative sign means that the VIX has diminished. Predicting correctly the sign means knowing if the VIX will increase or decrease on the next day.

We now present these three metrics for the in-sample data (training and validation data). For the series of logarithmic returns r_t:

	MSE	MAE	DA
ARIMA-GARCH	0.00372	0.0444	54.1%
Neural Network	0.00372	0.0444	53.4%
Hybrid	0.00369	0.0442	54.2%
AR(1)	0.00379	0.0448	51.1%

Table 4.3: r_t in-sample metrics for the selected models

The difference between the models is little but not minimal with the worst model that has a mean squared error about 3% larger than the best model. A difference that is practically significant is the directional accuracy that is 2.1 points higher for the best model with respect to the worst. On training data, the result is not unexpected, with the hybrid model as the best model on all metrics and the AR(1) as the worst. In this case, the complexity of the model seems to repay. It is interesting to note that the neural network alone performs worse than the ARIMA-GARCH.

For the series of first differences Δ_t:

	MSE	MAE	DA
ARIMA-GARCH	2.25	0.946	53.1%
Neural Network	2.22	0.949	52.1%
Hybrid	2.18	0.935	53.4%
AR(1)	2.29	0.948	51.1%

Table 4.4: Δ_t in-sample metrics for the selected models

Using the series of simple differences, the worst model is about 5% worse in term of MSE than the best model. This means that with a series less normal and with more extreme values, the complexity of a model helps to model the training data. Another time, the hybrid model is the best on all the metrics, in particular on the directional accuracy that is again 2.3 points higher than the accuracy of the AR(1). Surprisingly, the AR(1) has a mean absolute error that is slightly better than the one of the neural network. The difference is minimal, but show us a possible weak point of the single neural network.

4.3 Forecasts and Combination

On test data, a rolling forecast is performed to create the series of forecasts. At each time step, we use all the information until that time to predict the value of the next day. This means that we use the past values of the series as inputs of the ARIMA-GARCH and the neural network and the past forecast errors of the ARIMA-GARCH as inputs for the neural network of the hybrid model. The models are never re-estimated so we are using the same parameters in all the test period. This is not optimal if the interest is only the forecasting performance and the simple re-estimation of the models can be found useful. In this case, we want to test the generality in time of our models and see if it can be improved by a combination of forecasts.

The results of the one-day forecast on test data of r_t is presented:

	MSE	MAE	DA
ARIMA-GARCH	0.00697	0.0572	56.0%
Neural Network	0.00701	0.0573	54.2%
Hybrid	0.00697	0.0570	56.3%
AR(1)	0.00703	0.0581	50.8%

Table 4.5: r_t out-sample metrics for the selected models

On the test data, the models are again pretty similar for what concerns the errors while on the directional accuracy the ARIMA-GARCH and the hybrid model performs much better than the other two. The hybrid model is the best model among all metrics. It lowers the errors while keeping a directional accuracy much higher than the AR(1) and the neural network and a bit higher than the ARIMA. The worst model is the AR(1). The neural network is not able to overcome the ARIMA-GARCH in these logarithmically transformed data. It is worth notice that besides the fact that the neural network alone is overcome by the ARIMA-GARCH on all metrics, the subsequent application of a neural network after the ARIMA-GARCH to create the hybrid model works best. So the neural network approach seems to work better on the already filtered residuals tahn on the raw data.

The results for Δ_t are:

	MSE	MAE	DA
ARIMA-GARCH	4.28	1.142	53.4%
Neural Network	4.13	1.130	51.3%
Hybrid	4.14	1.124	53.4%
AR(1)	4.16	1.138	50.8%

Table 4.6: Δ_t out-sample metrics for the selected models

When modelling the less normal series, the situation changes. In this case, the worst model for mean squared error and mean absolute error is the ARIMA-GARCH that has larger errors than the one of the basic AR(1). This is surprising given that the selection process used for this ARIMA-GARCH is exactly the same used for the ARIMA-GARCH applied on r_t. This less normal series seems to be much harder to handle for the ARIMA-GARCH. The best is once again the hybrid model. This time the neural network beat the hybrid model on MSE but this is easily compensated by lower MAE and higher directional accuracy of the hybrid model. It is interesting to see that the models have much lower performances on forecasting the direction of the VIX for the next day when modelling the first difference series. Probably the presence of extreme values in some specific periods makes the models focus on modelling only these periods that are the greatest source of squared errors. The cost of a smaller squared error in these extreme moments is a loss on the general ability to understand the direction of the series. With this series, the hybrid model diminishes the errors of the ARIMA-GARCH. It can be interesting to understand how the hybrid model is acting to obtain this result. In the figure 4.3 the forecasting errors $y_t - \hat{y}_t$ of the ARIMA-GARCH model are plotted in the period right before the 2020 pandemic and in the first phase of the pandemic. As expected, the errors of the model increase greatly when the pandemic starts. The other line represents the correction applied by the neural network used in the hybrid model. This neural network tries to predict the errors of the ARIMA-GARCH. It is really interesting to see that the network apply almost no correction in the tranquil period before the pandemic. This suggests that the network is not able to improve further the forecasts of the ARIMA-GARCH in that period. When the pandemic starts, the neural networks ”activates” and start to apply correction to the forecasts of the ARIMA-GARCH. As a matter of fact, the MSE in the months of February and March 2020 is 55.6 for the ARIMA-GARCH and 51.2 for the hybrid model. Again, the difference is not huge, but the neural network attached to the ARIMA-GARCH improve by about 10% the MSE in a period of very high volatility. At the same time, the directional accuracy of the hybrid model in the same period is 46.3% while the one of the ARIMA-GARCH alone is 43.9%. A notable result for the hybrid model.

We do not provide the same plot for the series r_t because in that case the ARIMA-GARCH and the hybrid model do not show any difference in terms of MSE. In that case, with a series more normalized, they behave in a much similar way.

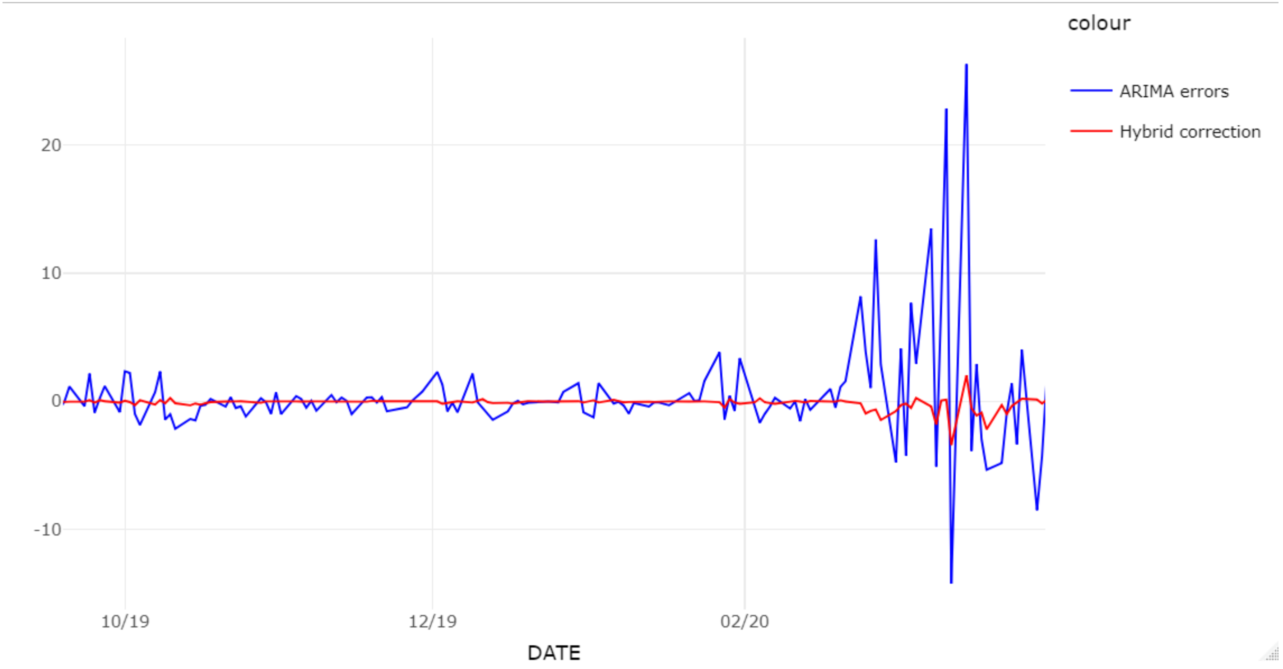


Figure 4.3: Plot of the forecasting errors of the ARIMA-GARCH model (blue) and the correction applied by the neural network used in the hybrid model (red) in the period 10/19 - 03/20

A comparison of the behaviour of the forecasts of the neural network and of the hybrid model can be found in Appendix C.

Once all the forecasts series have been built, we try the different combination methods we presented in Chapter 2. For the dynamic methods, a moving window of 500 observations is used to compute the optimal weight for the next time step. We present the results. In the combination are included the hybrid model, the neural network and the AR(1). For the logarithmic returns:

	MSE	MAE	DA
SA	0.00696	0.0573	56.0%
SREG	0.00699	0.0571	56.3%
DREG	0.00704	0.0577	54.8%
SUREG	0.00700	0.0570	56.5%
DUREG	0.00708	0.0579	53.3%
NN	0.00704	0.0576	53.4%

Table 4.7: r_t out-sample metrics of the combinations of forecasts

No combination seems to beat their single components on all metrics. The simple average combination is able to diminish the mean squared errors to a level slightly lower than the one of the best single component (hybrid model), but on the MAE and the DA is worse. An interesting result is obtained by the static unconstrained regression (SUREG) that was able to push a bit further the directional accuracy of the hybrid model keeping the same MAE and a very similar MSE. The DA is the metric most useful in real world and in real financial application, so a combination of models that increases it can be useful also if the increase is really small. In this case, the combination does not seems to be able to overcome the single hybrid model, but if we consider that the hybrid model is combined with worse models, it is notable how the combinations obtained results similar (or slightly better on some metrics) than the best model.

For the simple difference day on day:

	MSE	MAE	DA
SA	4.11	1.127	53.1%
SREG	4.17	1.127	53.2%
DREG	4.30	1.141	52.7%
SUREG	4.18	1.127	53.2%
DUREG	4.38	1.158	51.7%
NN	4.12	1.119	51.7%

Table 4.8: Δ_t out-sample metrics of the combinations of forecasts

Again we see that the simple average is able to give a minimal decrease to the MSE of all the models in the combination. The mean absolute error of the SA, the SREG and the SUREG is lower than the MAE of all the models except the hybrid model. Unfortunately, no combination was able to beat the hybrid model on directional accuracy, but the combination done with the neural network obtained lower MSE and MAE, resulting better than all the single models on error performance.

With both series does not seem that the combination of forecasts is able to give big advantages with respect to the hybrid model. In the plot 4.4 we show a 30 days rolling average of the MSE of the SA combination minus the MSE of the hybrid model for the Δ_t series. If the line is positive, the MSE of the SA combination has been higher than the one of the hybrid model in the previous 30 days. We clearly see that, even if the difference on the whole test data is very small, there are at least two periods where the MSE of the SA combination is visibly smaller. The first period is during the financial turbulence that started in August 2015 and the second is during the 2020 pandemic.

The simple average combination is then able to diminish the MSE during periods of high volatility, but this comes to the cost of a slightly reduced MAE and DA on the whole test dataset. Here we show only the results for the simple average combination because it proved to be the best combination for what concerns MSE on both r_t and Δ_t.

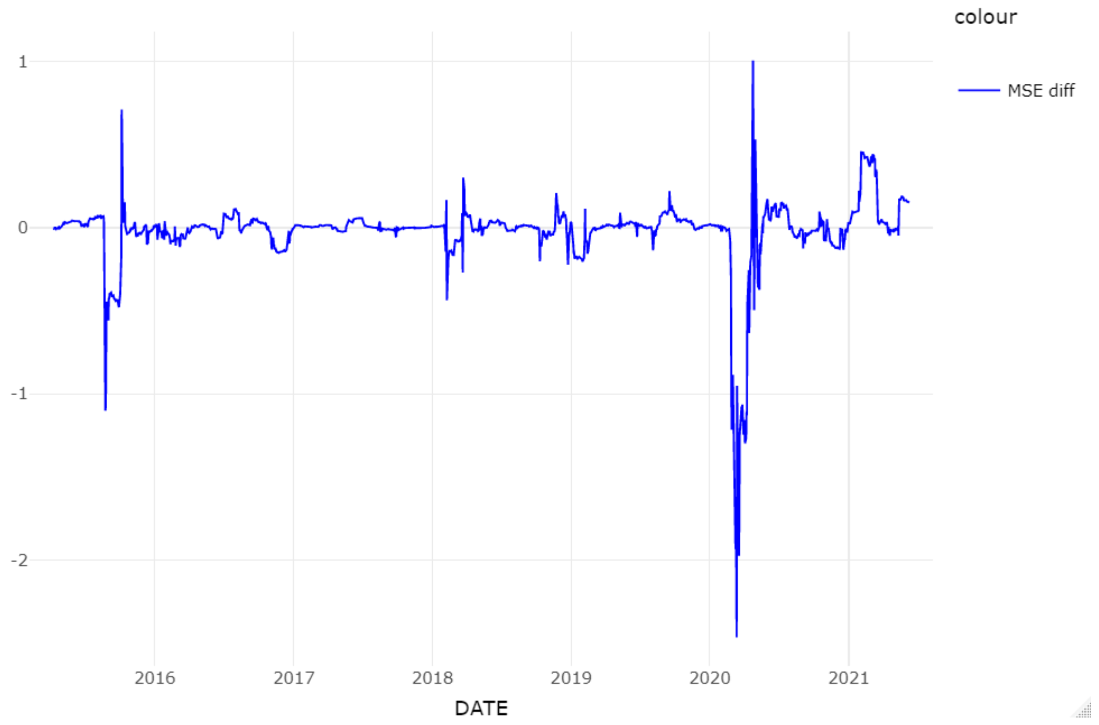


Figure 4.4: 30 days rolling mean of the difference between MSE of the SA combination and MSE of the hybrid model for Δ_t series

Chapter 5

Conclusions

The aim of this thesis was to propose a flexible procedure to extract the best characteristics of the ARIMA and of the feedforward neural network approach to time series. Our empirical results on the one-day forecast of the VIX shows that the hybrid model was consistently the best model in-sample and out-sample both on the first-difference series and on the log-returns series. The directional accuracy shown by the hybrid model is much larger than the one obtained by the neural network or the simple AR(1) model. The maximum directional accuracy was obtained in the series of logarithmic returns by the hybrid model. With a 56.3% accuracy, it is an improvement with respect to the previous works of Ahoniemi (2006) [3] and Konstantinidi et al. (2008) [30] (at least where no external regressors were used). If interested in predicting the direction of the VIX index, it seems advisable to help the models by normalizing the data by computing logarithmic returns. With respect to the ARIMA-GARCH, the hybrid models showed small improvements on all metrics, in particular, it was more able to handle very volatile data such as the ones of the 2020 pandemic crisis. It would not be a surprise to see the hybrid model to give larger improvements on a different set of data. The VIX seems to be pretty hard to predict and improvements over the ARIMA-GARCH were always present but not very large.

The combination of forecasts was not able to improve the results of the hybrid model on all metrics. This does not mean that it is useless to use combination in the real world. On both the series the simple average provided a little improvement of the mean squared error. Moreover, we have done some tests using ARIMA-GARCH models that were not the best according to our selection process. In these cases, the simple average was able to improve all three metrics with respect to the hybrid model done with a non-optimal ARIMA-GARCH. It is so our advice to keep using the simple average also when handling hybrid models. We showed that using the simple average can also partially protect from a sudden change of behaviour on the time series.

For what concerns the other combinations strategies, none of them was able to beat the simple average on mean squared error. While some shows improvements in some aspects with respect to the hybrid model, these improvements were limited to only one of the two series (r_t or Δ_t). So we do not find them to be very reliable. Moreover, the dynamic methods performed always worse than their static counterparts. Maybe this is due to the dimension of the moving window. We have tried different possibilities, from short to long windows, but we were not able to obtain good results. Of course, choosing the best window after having seen the data is not possible in real world application. It seems that in the VIX series, the past performances of a model are not very indicative of future performance.

Finally, even if the empirical results are not astonishing when compared to the classic ARIMA-GARCH, we hope that this thesis was useful to show a procedure applicable to a vast amount of different univariate time series to merge ARIMA and neural networks. We found the hybrid model to be the best overall, and this is in line with the findings of Zhang (2003) [49] and Tseng et al. (2002) [47]. These promising results suggest that the use of the hybrid approach should become standard as an alternative to the separate ARIMA or neural network models. We hope that the framework proposed by this thesis can be enough general to be useful also with other time series.

Appendix A

Appendix - R functions ARIMA selection

A.1 Roots check

The first function is used during the selection of the ARIMA-GARCH to check if the roots of the ARMA part have any clear problem. It receives as input an uGARCHfit object. Then it extracts from the coefficients vector all the parameters that correspond to the names ”ar” and ”ma”. It computes the roots of the characteristic polynomial both for AR and MA parameters. Then the plot of the inverse roots is drawn. Then, if the module of any root is less or equal to one, the boolean variable ”stationarity” is set to false. If there are roots of the MA part that are too near to the roots of the AR polynomial, a warning is given as output. All models with False stationarity are automatically discarded. All models with a warning should be analyzed by looking at the plot of the roots.

```
uroots=function(x){
  coef=x@fit$coef
  #Remove alla paramters with 'gamma'
  if ('gamma' %>% grepl(names(coef)) %>% any()) {
    coef=coef[-grep('gamma', names(coef))]
  }
  #Now we compute the roots
  ar=coef[grep('ar', names(coef))]
  ar.roots=polyroot(c(1,-ar))
  ma=coef[grep('ma', names(coef))]
  ma.roots=polyroot(c(1,-ma))

  #The plot of the inverse roots.
  plot(ar.roots^(-1), xlim=c(-1.5,1.5), ylim=c(-1.5, 1.5),
     asp=1, main='Inverse Roots')
  points(ma.roots^(-1), col='red', pch=2)
  legend("topleft", col=c("black","red"), legend=c("AR
     inverse Roots","MA inverse Roots"), pch=c(1,2))
  draw.circle(0,0, radius = 1, lty=3)
  plot=recordPlot()

  #Check stationarity
  if (any(Mod(ma.roots)<=1)|any(Mod(ar.roots)<=1)){
     stationarity=F} else {stationarity=T}

  #Check equal roots
  warn=NA
  for (i in ar.roots^(-1)) {
    if (any(near(Re(i), Re(ma.roots^(-1)), tol=0.1) & near(
       Im(i), Im(ma.roots^(-1)), tol=0.1))) {
      warn='Risk of equal root/s! Check the plot.'
    }
  }
  return(list(AR=ar.roots, MA=ma.roots, Stationarity=
     stationarity, warn=warn, plot=plot))
}
```

A.2 Significance check

The second function we wrote is used to automatically check the significance of all the parameters estimated in the ARIMA-GARCH. The function takes as inputs an uGARCHfit model and a level of significance (in the thesis we use 0.05). If any p-value is higher than the requested level, the function takes the parameter with the highest p-value and adds it to the list of the already fixed parameters of the model, if any. The updated list is then given as output. The code will estimate again the same model using the new list of fixed parameters. The process is repeated until the function returns the boolean variable ”sign” equal to True. This means that all the parameters are significant.

```r
usignificance = function(x, alpha){

  p.value=x@fit$matcoef[,4]  #compute pvalue

  #if any non significant coefficient, fix it to 0
  if (any(p.value>=alpha, na.rm=T)) {
    non.sign=which(p.value>=alpha)
    worst=p.value[non.sign][order(p.value[non.sign],
       decreasing = T)] %>% head(1)
    name=names(worst)
    fixed=x@model$fixed.pars
    fixed$temp=0
    names(fixed)[length(fixed)]=name

    sign=F
  } else {
    fixed=x@model$fixed.pars
    sign=T
  }
  return(list(sign=sign,fixed=fixed))
}
```

Appendix B

Appendix - ARIMA-GARCH parameters and diagnostics

B.1 Logarithmic returns series

The model selected for the r_t series is an ARMA(1,5)-apARCH(1,1) that uses a skewed Student-t distribution as conditional distribution for the standardized residuals and has 3 parameters fixed to zero due to non-significance (the constant for the conditional mean and the parameters for the second and fourth order of MA). Here we present the estimation of parameters:

```
--------------------------------------------------
         Estimate  Std. Error      t value  Pr(>|t|)
mu       0.000000          NA           NA        NA
ar1      0.971998    0.002550   3.8117e+02  0.000000
ma1     -1.068845    0.000000  -2.2668e+06  0.000000
ma2      0.000000          NA           NA        NA
ma3      0.026633    0.005230   5.0926e+00  0.000000
ma4      0.000000          NA           NA        NA
ma5      0.049417    0.005285   9.3506e+00  0.000000
omega    0.003382    0.001011   3.3448e+00  0.000823
alpha1   0.070875    0.006252   1.1336e+01  0.000000
beta1    0.859670    0.013775   6.2407e+01  0.000000
gamma1  -1.000000    0.000352  -2.8408e+03  0.000000
```

```
delta     1.121562     0.102515    1.0940e+01  0.000000
skew      1.268864     0.022100    5.7414e+01  0.000000
shape     6.409764     0.488347    1.3125e+01  0.000000
----------------------------------------------------------
```

All non-restricted parameters are highly significant. ω, α_1 and β_1 are positive, as requested for a correct GARCH specification. The γ of the apARCH model is equal to -1. This means that the conditional volatility is influenced only by positive ε_t. The negative ones are deleted. δ is different from the standard 2 used in the vanilla GARCH. The skew parameter is different from 1, so the conditional distribution of the standardized residuals is skewed.

The diagnostics plots are shown:

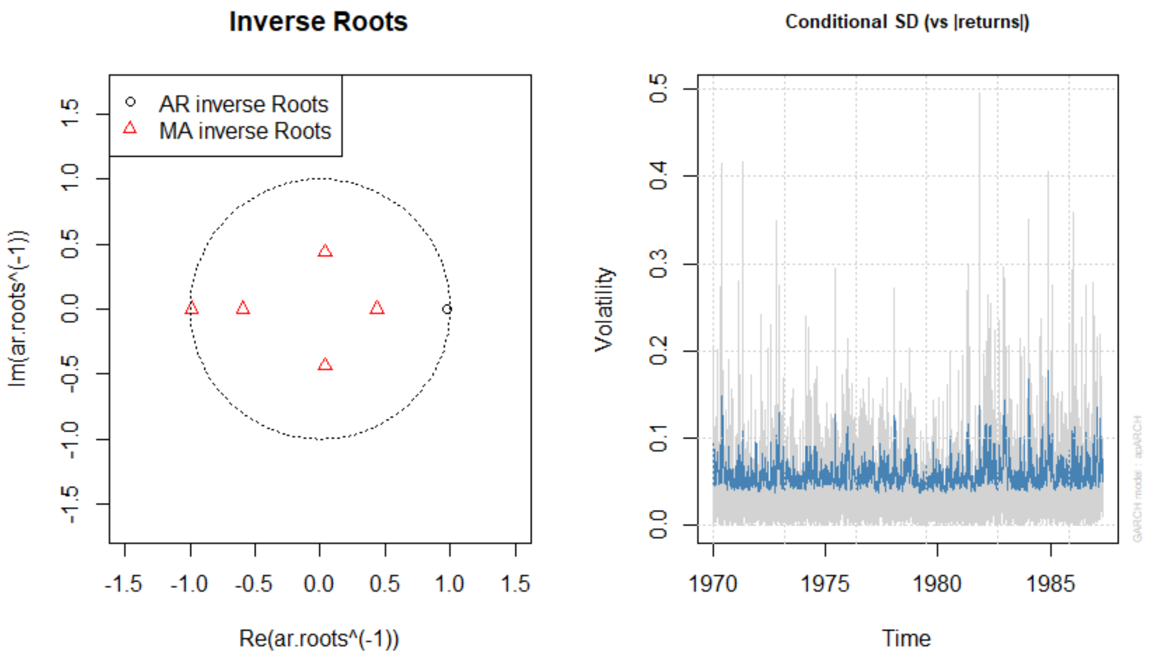


Figure B.1: Left: Inverse roots of the ARMA polynomials. Right: fitted conditional SD compared with actual absolute values of the series

The roots of the AR and the MA characteristics polynomials have all module greater than 1. In the plot of the inverse roots, we see that two roots are very near to module 1, while staying anyway inside the unit circle. The conditional standard deviation does not seem to be able to catch all the spikes in the absolute values series.

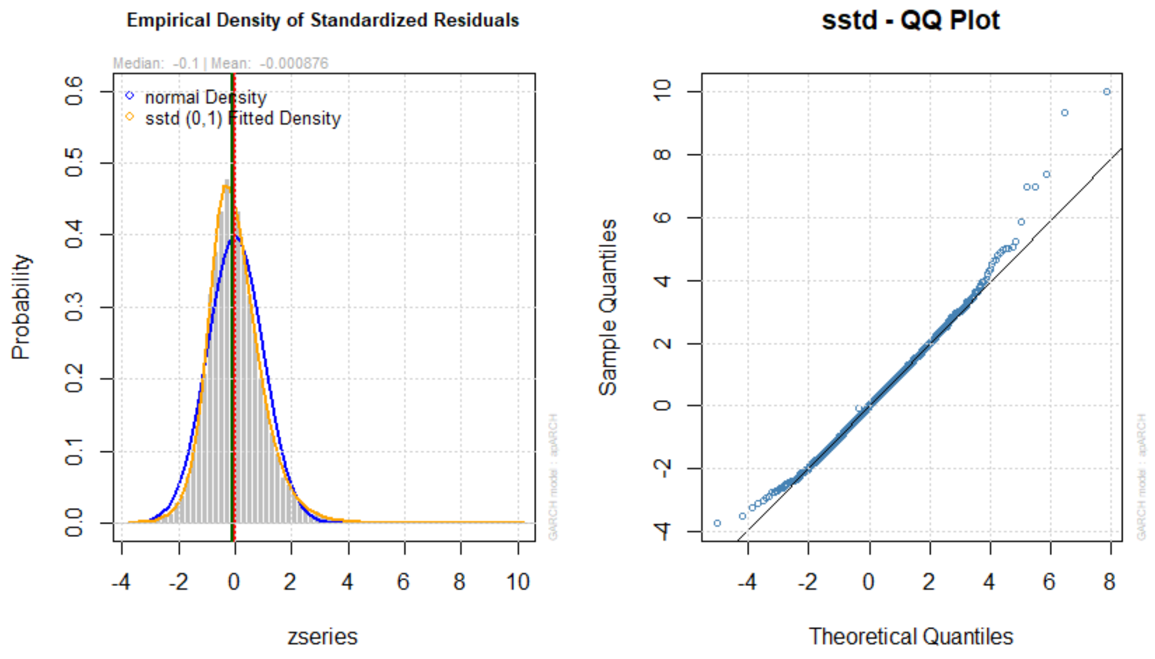


Figure B.2: Left: Theoretical distribution (orange) compared with empirical desnity of standardized residuals. Right: Theoretical quantiles compared with empirical ones

The theoretical distribution seems to be pretty good in approximating the distribution of the standardized residuals. We can see that the distribution is not much different from the standard normal. This is because the logarithmic transformation helps the data to be more similar to a normal. The Adjusted Pearson Goodness-of-Fit test confirms with very high p-values the right choice of the skewed Student-t distribution.

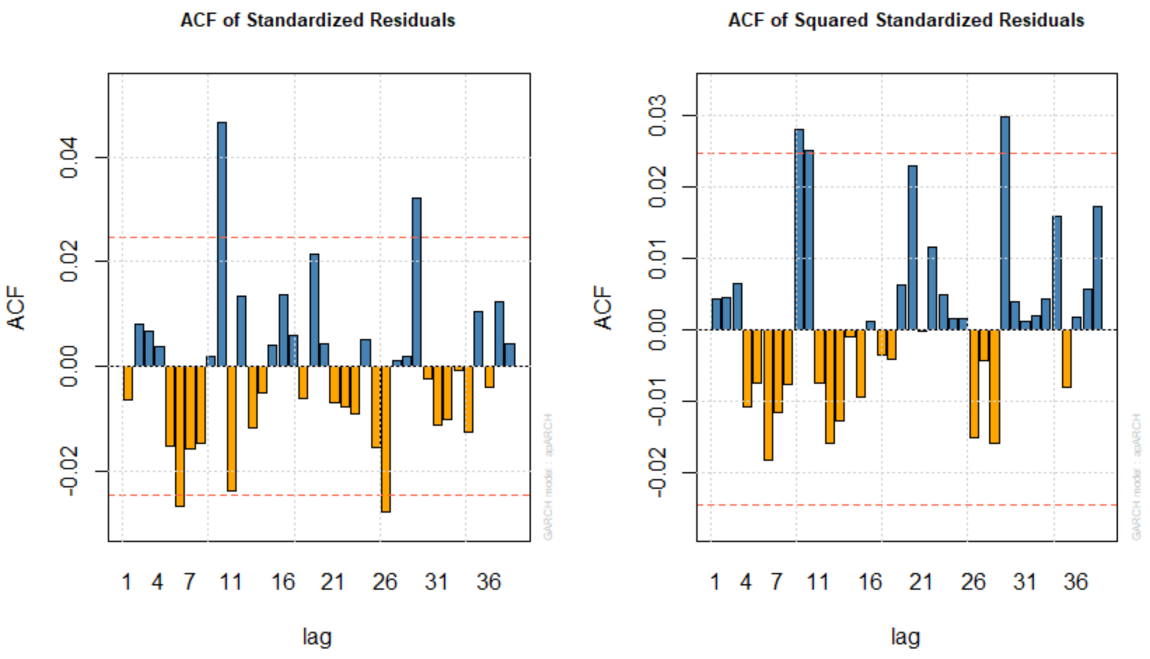


Figure B.3: Left: ACF standardized residuals. Right: ACF squared standardized residuals

The standardized residuals do not show any residuals autocorrelation for the first 5 lags. This is the most important thing for our selection that tests all models with a maximum order of 5. With a greater number of lags, some correlation is significant. It would be optimal to try all the models until a very high order and then choose the best. Of course this is much more heavy from a computational point of view. The same facts are true for the autocorrelation of the standardized residuals.

B.2 First difference series

The model selected for the first difference series Δ_t is an ARMA(2,1)-apARCH(1,1) with a skewed Student-t for standardized residuals and no parameters fixed to zero. Here we provide the estimation of the parameters:

```
--------------------------------------------------
        Estimate   Std. Error    tvalue   Pr(>|t|)
mu      0.027918     0.005009    5.5731   0.000000
ar1     0.615849     0.087898    7.0064   0.000000
ar2    -0.039707     0.016437   -2.4157   0.015706
ma1    -0.719266     0.078592   -9.1519   0.000000
```

```
omega      0.014727    0.004523         3.2561    0.001130
alpha1     0.083087    0.009234         8.9982    0.000000
beta1      0.937622    0.013264        70.6866    0.000000
gamma1    -0.999999    0.000067    -15027.8763    0.000000
delta      0.826143    0.100635         8.2093    0.000000
skew       1.332560    0.021598        61.6982    0.000000
shape      5.143769    0.332990        15.4472    0.000000
-----------------------------------------------------------
```

All the parameters are significant for a level of 0.05. ω, α_1 and β_1 are positive, as we expect for a correct GARCH model. The γ parameter is different from zero and the δ is different from 2, so the apARCH does not reduce to a standard GARCH model. The leverage parameter γ is extremely near to -1. This means that the conditional volatility is increased only by positive errors. The negative errors account zero to the increase of conditional volatility. The skew parameter is different from 1, so the distribution of the standardized residuals is skewed.

The plots used for diagnosis are presented:

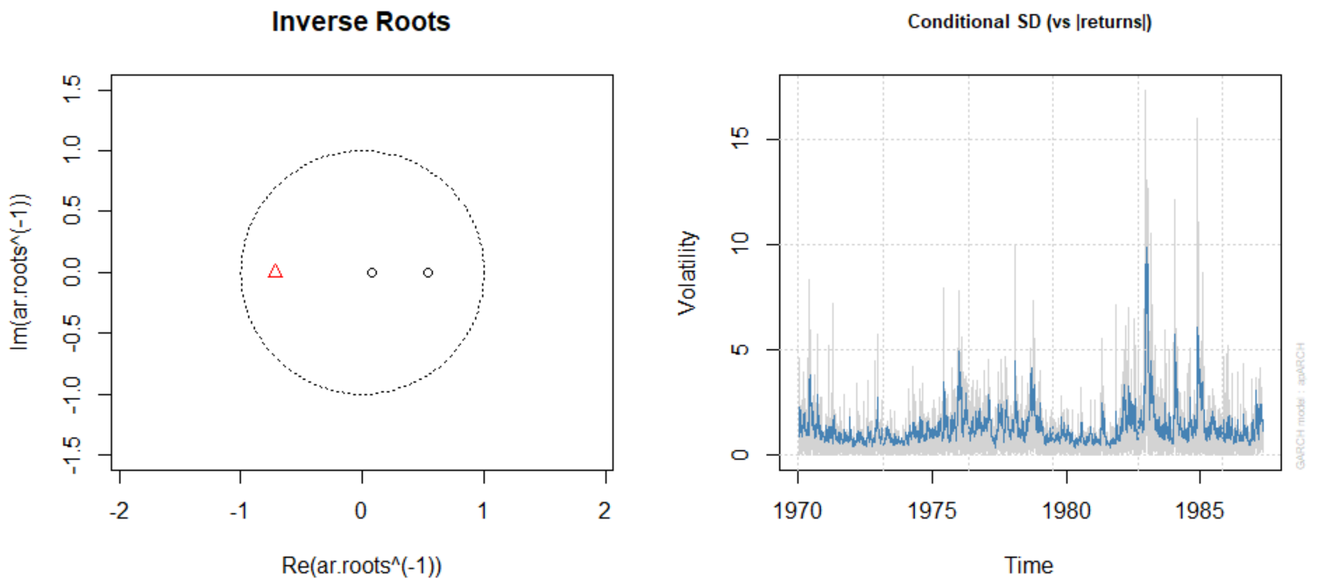
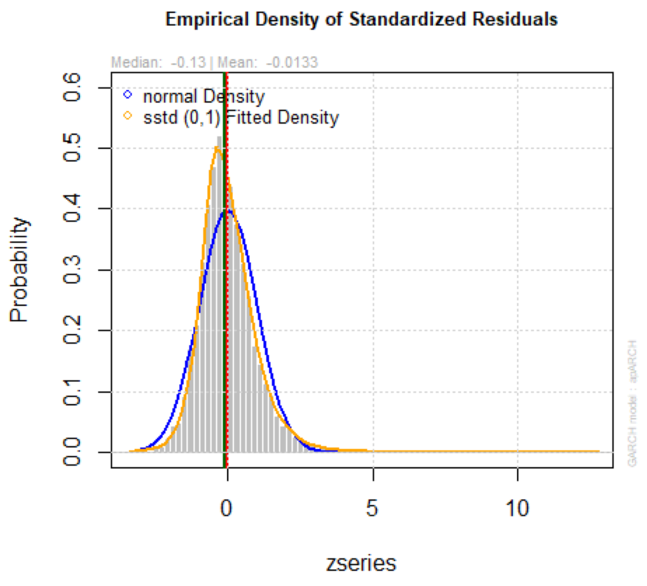


Figure B.4: Left: Inverse roots of the ARMA polynomials. Right: fitted conditional SD compared with actual absolute values of the series

From figure B.4 we see that the inverse roots of the MA and AR characteristic polynomials are inside the unit circle. So the roots not inverted are outside the unit circle confirming the stationarity and invertibility of the process. The conditional standard deviation is able to model pretty well the spikes in the absolute value series.



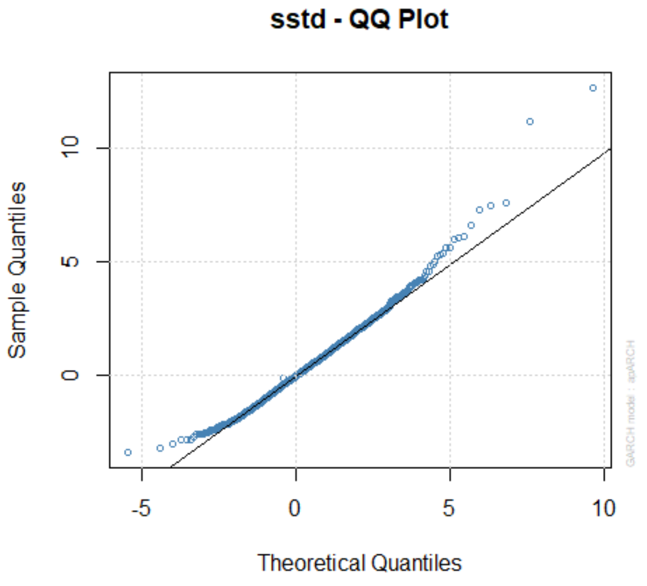
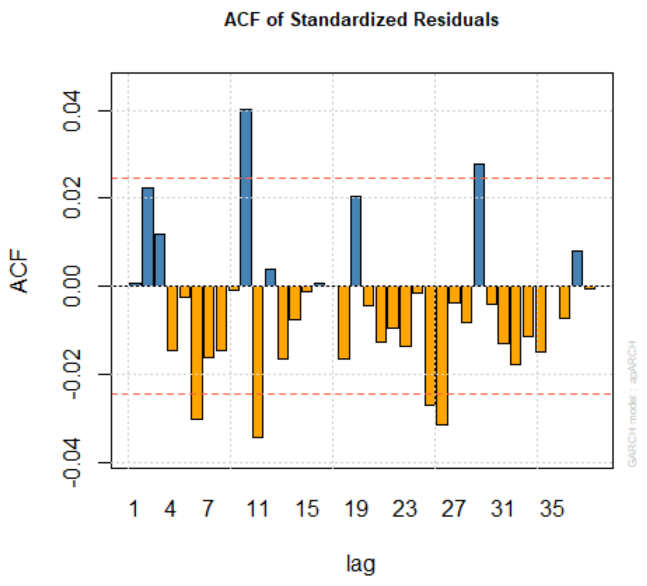


Figure B.5: Left: Theoretical distribution (orange) compared with empirical desnity of standardized residuals. Right: Theoretical quantiles compared with empirical ones

The chosen distribution (skewed Student-t) seems to describe well the distribution of the residuals of the model. This is confirmed by an Adjusted Pearson Goodness-of-Fit Test that results in very high p-values with different numbers of bins.



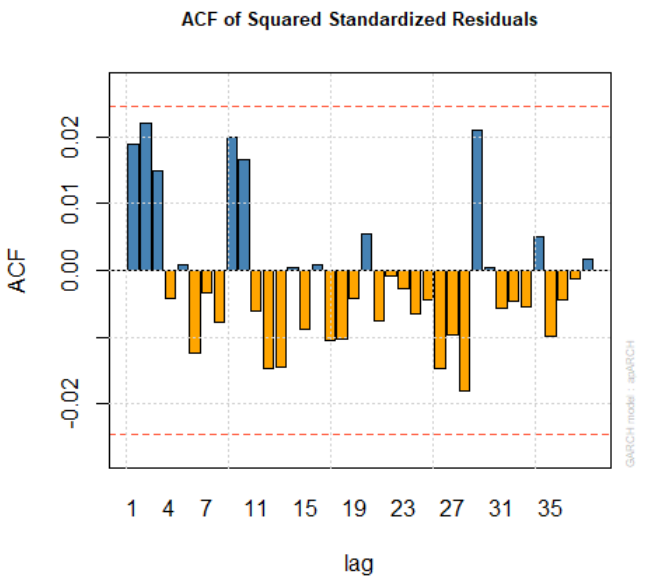


Figure B.6: Left: ACF standardized residuals. Right: ACF squared standardized residuals

The model is able to capture the linear correlation of the first five lags. This is of course the most important thing given that we selected the best models testing all models until a maximum ARMA order of (5,5). So the selected model is effectively able to handle the correlation with the first 5 lags, using only an ARMA(2,1). Increasing the lag, we see that some correlation is present. It would be optimal to test all the models until a higher order. This is much more computationally heavy but should improve our model. The squared standardized residuals do not show any residual autocorrelation.

Appendix C

Appendix - Comparison Hybrid model and Neural Network

C.1 Logarithmic returns series

Now we show a comparison of the performance of the neural network and of the hybrid model on the series r_t:

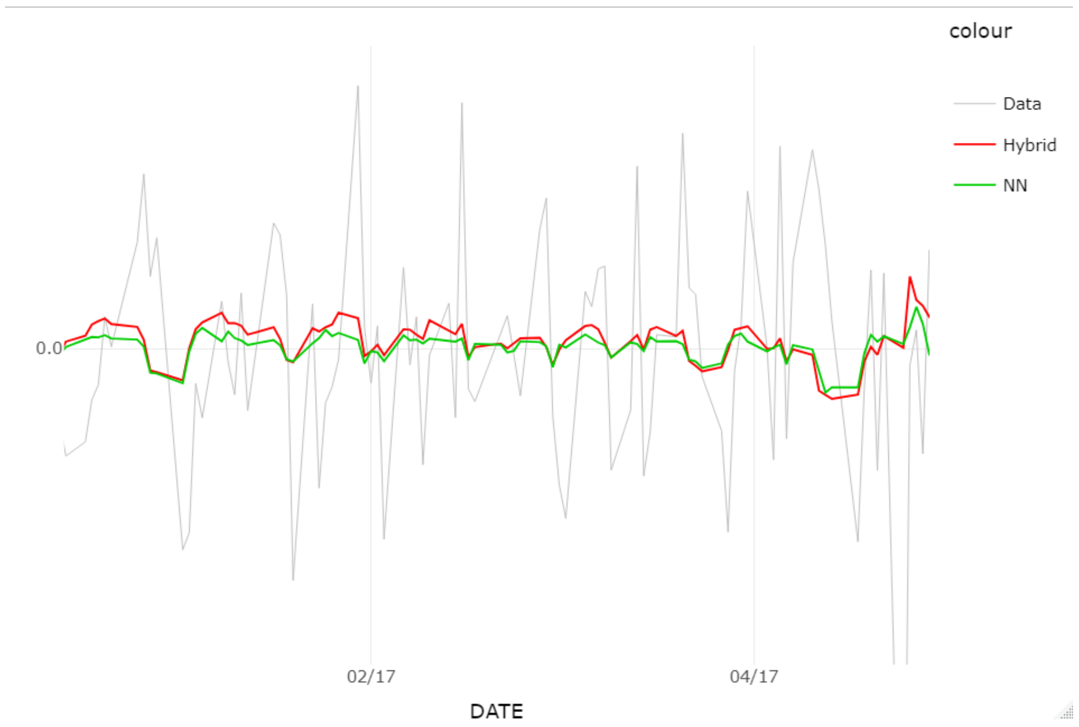


Figure C.1: Actual data, forecasts of the hybrid model (red) and forecasts of the neural network (green) for the r_t series. January 2017 - April 2017

In the plot C.1 we see that the two series of forecasts have very similar behaviour.

The example provided is in a tranquil period. The only difference stays in the fact that the hybrid models shows a bit more variance than the neural network. This fast change of direction can be the basis for the ability of the hybrid model to predict the direction of the series better than the neural network. This suspect is confirmed by the next plot that shows how the forecasts provided by the hybrid model are much more spread than what is obtained by the neural network alone. The standard deviation of the forecasts of the neural network is 0.014 while the one of the hybrid model is 0.016.

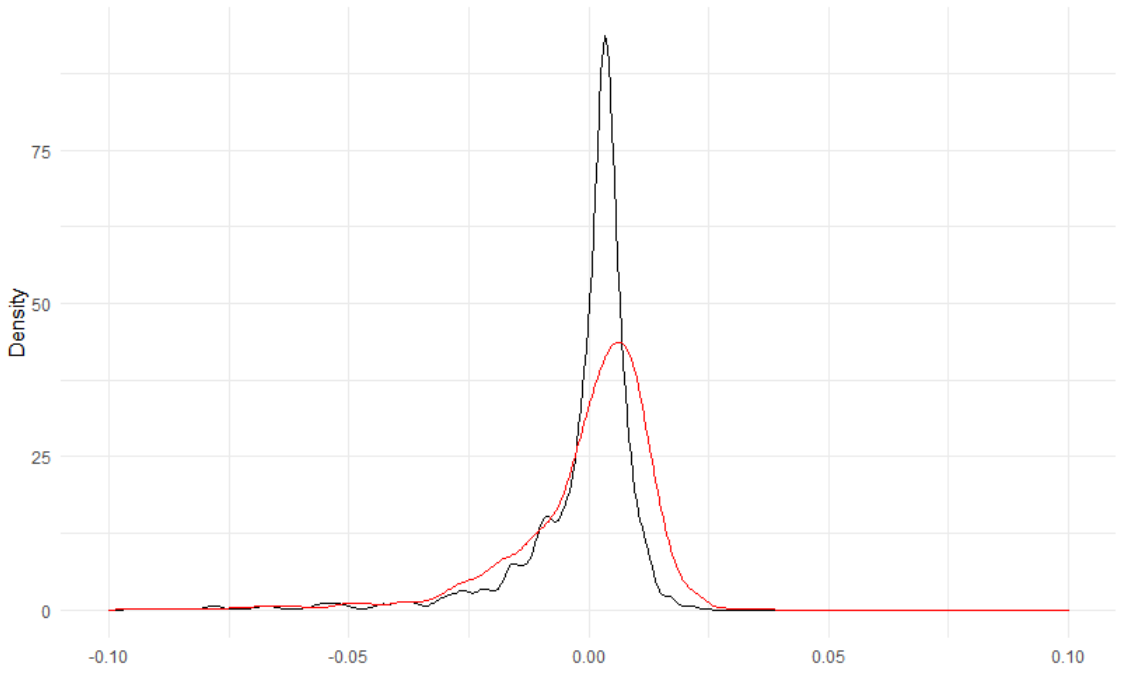


Figure C.2: Kernel density of the forecasts of the neural network (black) and of the hybrid model (red)

C.2 First difference series

We compare the neural network and the hybrid model on the series Δ_t. In the plot C.3 we show the two series of forecasts of the hybrid model and of the neural network for an example subset of test data. The period shown is a tranquil period. It is clear that the hybrid model has a higher variance while the Neural Network is much more stable around the mean. This can explain why the neural hybrid model is able to achieve a much better directional accuracy than the single neural network. In plot C.4 we provide the kernel density estimation of the forecasts of both models. The forecasts of the neural network are much more concentrated than the forecasts of the hybrid model. The standard deviation of the forecasts of the neural network is 0.36 while for the hybrid model is 0.47.

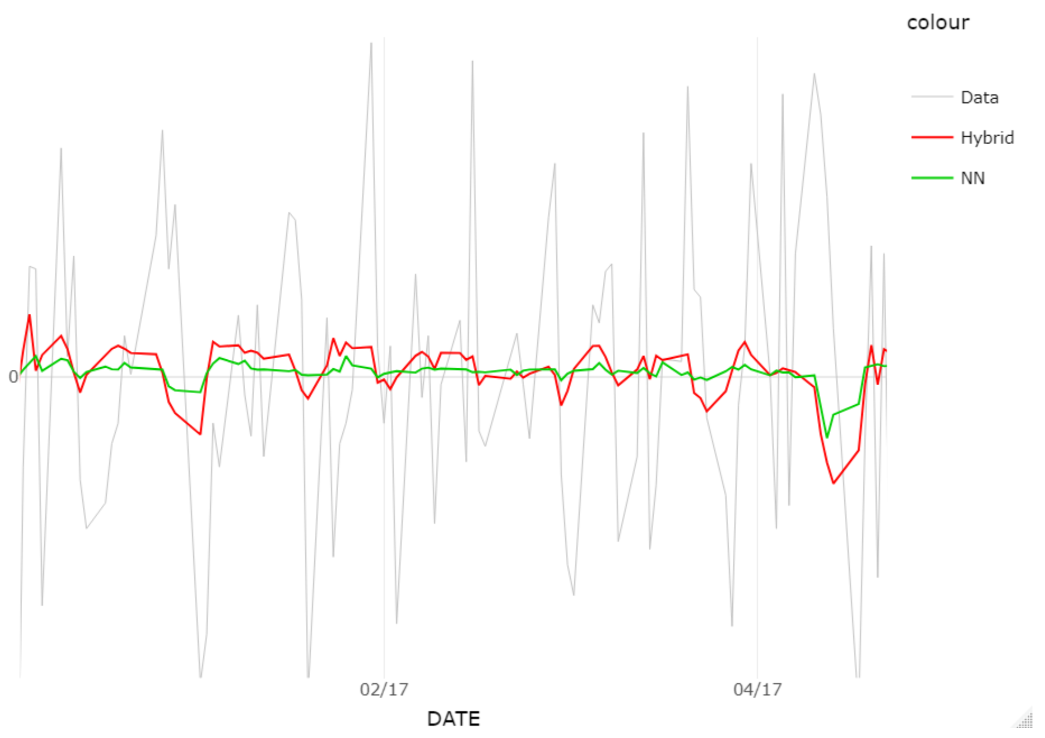


Figure C.3: Actual data, forecasts of the hybrid model (red) and forecasts of the neural network (green) for the Δ_t series. January 2017 - April 2017

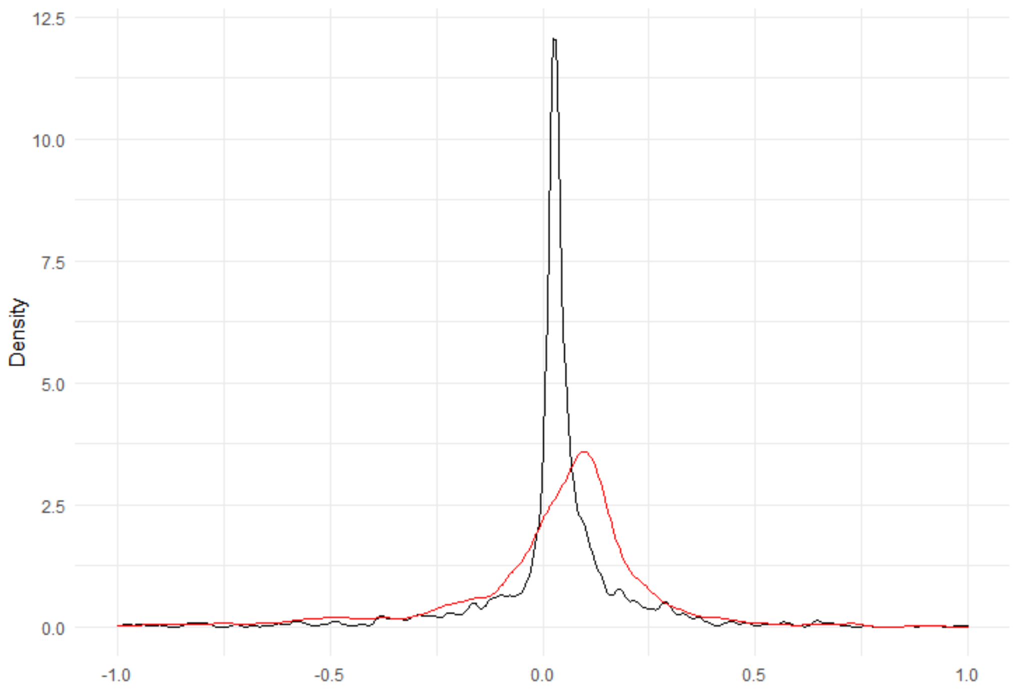


Figure C.4: Kernel density of the forecasts of the neural network (black) and of the hybrid model (red)

Bibliography

[1] Abu-Mostafa Y.S., Atiya A.F. (1996), Introduction to financial forecasting, Appl Intell 6, 205–213

[2] Adhikari R. and Agrawal R. K. (2013), An Introductory Study on Time Series Modeling and Forecasting, LAP Lambert Academic Publishing, Germany

[3] Ahoniemi, K. (2006), Modeling and forecasting implied volatility: An econometric analysis of the VIX index, Working paper, Helsinki School of Economics.

[4] Aladag C.H., Egrioglu E., Yolcu U. (2010), Forecast Combination by Using Artificial Neural Networks, Neural Process Lett, 32:269–276

[5] Bates J. M. and Granger C. W. J. (1969), The Combination of Forecasts , OR Vol. 20, No. 4, pp. 451-468

[6] Bishop C. M. (2006), Pattern Recognition and Machine Learning, Springer, Aug 17, 2006

[7] Blanc S.M., Setzer T. (2016), When to choose the simple average in forecast combination, Journal of Business Research, Volume 69, Issue 10, Pages 3951-3962

[8] Box G.E.P. (1976), Science and Statistics, Journal of the American Statistical Association , Vol. 71, No. 356, pp. 791-799

[9] Box G. E. P., Jenkins G. M.,Reinsel G. C., Ljung G. M. (2015), Time Series Analysis: Forecasting and Control, 5th Edition, Wiley

[10] Brenner M. and Galai D. (1989), New Financial Instruments for Hedge Changes in Volatility, Financial Analysts Journal, 45:4, 61-65

[11] Burnham, K. P.; Anderson, D. R. (2002), Model Selection and Multimodel Inference: A practical information-theoretic approach (2nd ed.), Springer-Verlag.

[12] Cboe Exchange (2019), VIX White Paper, https://cdn.cboe.com/resources/vix/vixwhite.pdf

[13] Clemen R.T. (1989), Combining forecasts: A review and annotated bibliography, International Journal of Forecasting 5, 559-583, North-Holland

[14] Danielsson J. (2011), Financial Risk Forecasting: The Theory and Practice of Forecasting Market Risk with Implementation in R and Matlab , John Wiley & Sons

[15] Degiannakis S.A. (2008), Forecasting VIX, Journal of Money, Investment and Banking, ISSN 1450-288X Issue 4

[16] Ding Z. , Granger C. W. J.and Engle R. F. (1993), A long memory property of stock market returns and a new model, Journal of Empirical Finance, 83-106. North-Holland

[17] Donaldson R.G., Kamstra M. (1996), Forecast Combining with Neural Networks, Journal 01 Forecasting, Vol. 15,49-61

[18] Fernandez C. and Steel M.F. (1998), On bayesian modeling of fat tails and skewness, Journal of the American Statistical Association, 93(441):359–371

[19] Ghalanos A. (2020), Introduction to the rugarch package (Version 1.4-3)

[20] Glorot X., Bordes A., Bengio Y (2011), Deep Sparse Rectifier Neural Networks, Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, PMLR 15:315-323

[21] Granger C.W.J., Ramanathan R. (1984), Improved Methods of Combining Forecasts, Journal of Forecasting, Vol. 3, 197-204

[22] Granger C. W. J. (1989), Invited Review Combining Forecasts-Twenty Years Later, Journal of Forecasting, Vol. 8, 167-173

[23] Mitchell T. M. (1997), Machine Learning, McGraw-Hill Science/Engineering/Math

[24] Green K.C., Armstrong J.S. (2015), Simple versus complex forecasting: The evidence, Journal of Business Research 68, 1678–1685

[25] Harrald P.G. and Kamstra M. (1997), Evolving Artificial Neural Networks to Combine Financial Forecasts, IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 1, NO. 1

[26] Hinton G. E., Srivastava N., Krizhevsky A., Sutskever I., Salakhutdinov R. (2012), Improving neural networks by preventing co-adaptation of feature detectors,arXiv, arXiv:1207.0580

[27] Hornik K. (1991), Approximation capabilities of multilayer feedforward networks, Neural Networks, Volume 4, Issue 2, Pages 251-257

[28] Kingma D. P. and Ba J. (2015), Adam: a Method for Stochastic Optimization , International Conference on Learning Representations, pages 1–13

[29] Kolarik T. and Rudorfer G. (1994), Time series forecasting using neural networks, SIGAPL APL Quote Quad 25, 1 86–94

[30] Konstantinidi E., Skiadopoulos G., Tzagkaraki E. (2008), Can the evolution of implied volatility be forecasted? Evidence from European and US implied volatility indices, Journal of Banking & Finance, Volume 32, Issue 11, Pages 2401-2411

[31] LeCun Y.A., Bottou L., Orr G.B., Müller KR. (2012), Efficient BackProp, In: Montavon G., Orr G.B., Müller KR. (eds) Neural Networks: Tricks of the Trade, Lecture Notes in Computer Science, vol 7700, Springer, Berlin, Heidelberg

[32] LeCun, Y., Bengio, Y. and Hinton, G. (2015), Deep learning, Nature 521, 436–444

[33] Leshno M., Lin V. Ya., Pinkus A., Schocken S. (1993), Multilayer feedforward networks with a nonpolynomial activation function can approximate any function, Neural Networks, Volume 6, Issue 6, Pages 861-867

[34] Liang S. and Srikant R. (2017), Why Deep Neural Networks for Function Approximation?, arXiv

[35] de Menezes L.M., Bunn D.W., Taylor J.M. (2000), Review of guidelines for the use of combined forecasts, European Journal of Operational Research, Volume 120, Issue 1, Pages 190-204

[36] Mohammadi H. and Su l. (2010), International evidence on crude oil price dynamics: Applications of ARIMA-GARCH models, Energy Economics Volume 32, Issue 5, Pages 1001-1008

[37] Nelson D. B. (1991), Conditional Heteroskedasticity in Asset Returns: A New Approach, Econometrica Vol. 59, No. 2, pp. 347-370 (24 pages)

[38] Ruder S. (2016), An overview of gradient descent optimization algorithms, ArXiv

[39] Osterrieder J., Kucharczyk D., Rudolf S., Wittwer D. (2020), Neural networks and arbitrage in the VIX, Digital Finance, 2:97–115

[40] Psaradellis I., Sermpinis G. (2016), Modelling and trading the U.S. implied volatility indices. Evidence from the VIX, VXN and VXD indices, International Journal of Forecasting, 32, pp. 1268-1283

[41] Sezer O. B., Gudelek M. U., Ozbayoglu A. M. (2020), Financial time series forecasting with deep learning : A systematic literature review: 2005–2019, Applied Soft Computing, Volume 90, 106181

[42] Shakhnarovich G. (2011), Notes on derivation of bias-variance decomposition in linear regression

[43] Srivastava N., Hinton G., Krizhevsky A., Sutskever I., Salakhutdinov R. (2014), Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Journal of Machine Learning Research

[44] Svozil D., Kvasnicka V., Pospichal J. (1997), Introduction to multi-layer feed-forward neural networks, Chemometrics and Intelligent Laboratory Systems 39 43-62

[45] Szandała T. (2020), Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks, arXiv, arXiv:2010.09458

[46] Tang Z. and Fishwick P. A. (1993), Feedforward Neural Nets as Models for Time Series Forecasting, INFORMS Journal on Computing, INFORMS, vol. 5(4), pages 374-385

[47] Tseng F., Yu H., Tzeng G. (2002), Combining neural network model with seasonal time series ARIMA model, Technological Forecasting and Social Change, Volume 69, Issue 1

[48] Ye, Y. (1987), Interior Algorithms for Linear, Quadratic, and Linearly Constrained Non-Linear Programming, PhD Thesis, Department of ESS, Stanford University, Stanford

[49] Zhang G.P. (2003), Time series forecasting using a hybrid ARIMA and neural network model, Neurocomputing, Volume 50

[50] Zhou B., He D., Sun Z. (2006), Traffic Modeling and Prediction using ARIMA/-GARCH Model , In: Nejat Ince A., Topuz E. (eds) Modeling and Simulation Tools for Emerging Telecommunication Networks, Springer, Boston