Neurocomputing 122 (2013) 501–512

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Time-dependent series variance learning with recurrent mixture density networks

Nikolay Nikolaev [a,*], Peter Tino [b], Evgueni Smirnov [c]

[a] Department of Computing, Goldsmiths College, University of London, London SE14 6NW, UK
[b] School of Computer Science, The University of Birmingham, Birmingham B15 2TT, UK
[c] Department of Knowledge Engineering, Faculty of Humanities and Science, Maastricht University, Maastricht 6200, MD, The Netherlands

ARTICLE INFO

Article history:
Received 24 July 2012
Received in revised form
15 May 2013
Accepted 21 May 2013
Communicated by B. Hammer
Available online 10 June 2013

Keywords:
Mixture density neural networks
GARCH models
Real-time recurrent learning algorithm

ABSTRACT

This paper presents an improved nonlinear mixture density approach to modeling the time-dependent variance in time series. First, we elaborate a recurrent mixture density network for explicit modeling of the time conditional mixing coefficients, as well as the means and variances of its Gaussian mixture components. Second, we derive training equations with which all the network weights are inferred in the maximum likelihood framework. Crucially, we calculate temporal derivatives through time for dynamic estimation of the variance network parameters. Experimental results show that, when compared with a traditional linear heteroskedastic model, as well as with the nonlinear mixture density network trained with static derivatives, our dynamic recurrent network converges to more accurate results with better statistical characteristics and economic performance.

1. Introduction

The dynamics of the time-dependent variance in series of returns on prices, known also as the volatility, is of particular interest in finance as it impacts the pricing of financial instruments, and it is a key concept in market regulation. A popular tool for capturing the volatility are the Generalized Autoregressive Conditional Heteroskedastic (GARCH) models [7,15]. The current research studies volatility models with various non-normal return distributions [4,13], and flexible non-linear models [19,20,25,30].

The research into flexible models proposed non-linear GARCH based on neural networks [5,14]. However, this approach to volatility modeling using feed-forward neural networks cannot handle general spatial–temporal information (beyond finite input memory), and does not propagate through time gradient training information. Temporal dimension was added to non-linear GARCH with the recurrent density network (RDN) [18]. The RDN network is driven by external inputs as well as by internal context signals (dynamic state) that carry temporal information. The motivation for using RDNs for volatility modeling comes from the following advantages of recurrent networks: (1) they can learn time-varying distributions as they explicitly describe heteroskedastic dependencies in time series data; (2) they allow accommodation of non-linearities through the use of different activation functions; and (3) they enable computation of analytical temporal derivatives, and the implementation of dynamic learning procedures.

Recurrent mixture density networks (RMDN) [27,32,33] were designed to capture general non-Gaussian density specifications of 'arbitrary' shapes using mixtures of Gaussians [6,28]. Such an approach has been also applied to design mixtures of linear GARCH in order to account for non-normality in returns [11,21,34]. Normal mixture densities help to approximate the skewness and excess kurtosis in time series data, and to achieve better fit compared to single normal and Student-*t* densities [21]. Although these mixture models account for spatiotemporal information, they still infer the parameters with static algorithms and do not treat directly the temporal dimension of the volatility model during training. The key idea for making RMDN to represent mixture GARCH should be to exploit the recurrent networks as dynamic machines that learn time-dependent functions [36,37,38]. During training the parameters should be adapted in response to both the inputs and the internal temporal context of stored activations from previous time steps. This will turn the model into a dynamic function that reflects the time-dependencies in the data.

This paper proposes an improved RMDN-GARCH network for time-varying conditional density estimation in the general case where the "emission model" is assumed to be a mixture of Gaussians. The developed RMDN-GARCH architecture and dynamic training algorithm provide a universal non-linear formalism that can learn parameters in non-Gaussian distributions (captured by Gaussian mixtures) with analytical update formulas.

* Corresponding author. Tel.: +44 2079197854.
E-mail addresses: n.nikolaev@gold.ac.uk (N. Nikolaev), P.Tino@cs.bham.ac.uk (P. Tino),Smirnov@maastrichtuniversity.nl (E. Smirnov).

0925-2312/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.neucom.2013.05.014

There are two main contributions in this research: (1) the mixture network architecture, including its models for the mixing coefficients, means and variances, is elaborated to specify separately the linear and the non-linear terms (that is why the name RMDN-GARCH is taken to denote close correspondence to GARCH), this helps to tune more accurately the model to the data; and (2) we formulate original temporal derivatives of the likelihood of the mixture density network with respect to the recurrent variance parameters to describe accurately the time-relationships in the data. We derive dynamic likelihood gradients with respect to the weights in the variance network, representing the mean, the persistence and the moving average coefficients in the model. The rationale for having dynamic derivatives is to facilitate the convergence of the training process when used for learning by optimization.

The temporal derivatives of the likelihood function include terms computed following the real time recurrent learning (RTRL) training algorithm [37,38] for recurrent networks, which not only updates the weights but also memorizes temporal information generated by the network node activations. Thus, the overall model is dynamic and reacts to the current inputs as well as to its internal network context (state) at a specific time step. The proposed here RTRL algorithm yields temporal derivatives of the log-likelihood function that are a generalization of the analytical (closed-form) derivatives [16] for accurate estimation of linear GARCH models.

Empirical investigations were conducted with the following objectives: (1) to examine the influence of the network architecture and dynamic training on mixture density time series modeling; and (2) to compare linear GARCH with RMDN-GARCH and other recurrent density networks. First, we trained the models on simulated series and show that our RMDN-GARCH learns more accurate models with lower average fitting errors compared to the previous RMDN [33]. Second, we applied the RMDN-GARCH to real-world benchmark series and related its performance to nonlinear recurrent density networks and linear GARCH estimated by maximum-likelihood using analytical derivatives [16] and Monte Carlo MCMC sampling [24]. It was found that RMDN-GARCH leads to results with better statistical characteristics and better average out-of-sample economic performance than the other models. Specifically, the dynamic RMDN-GARCH trained using RTRL derivatives within a BFGS optimizer outperformed RMDN trained using static backpropagation derivatives [33] as well as using static derivatives computed by numerical differencing. Finally versions using Student-t noise were evaluated to demonstrate behavior under different distributional assumptions.

The remainder of this paper is organized as follows. Section 2 introduces the structure of nonlinear mixture GARCH models, including our mixture density RMDN-GARCH network. Section 3 gives the likelihood derivatives, including the novel temporal derivatives, and their use in the optimization algorithm for the mixture density network. Section 4 presents experimental results in a volatility inference task. Finally, we provide a discussion and conclusion in Section 5.

2. Nonlinear mixture GARCH modeling

2.1. The GARCH(p,q) model

The volatility of returns on assets vary over time, and it is modeled by an unobserved process of time-changing variance. Consider the log-returns from a series of asset prices $S_t, 1 \le t \le T$, that is $r_t = \log(S_t/S_{t-1})$.

According to the generalized autoregressive conditional heteroskedastic (GARCH)(p,q) model [7], the dynamics of log-returns on prices is described by the following equations:

$$r_t = \mu_t + e_t \tag{1}$$

$$\mu_t = a_0 + a_1 r_{t-1} \tag{2}$$

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i e_{t-i}^2 + \sum_{j=1}^{p} \beta_j \sigma_{t-j}^2 \tag{3}$$

where μ_t and σ_t are the mean and volatility of the returns distribution, $e_t \sim \mathcal{N}(0, \sigma_t^2)$ are zero-mean normal random variables of variance σ_t^2, a_0 is a constant (bias), and a_1 is an autoregressive coefficient.

Let us denote the series of all past returns arrived up to time t by $R_t = (r_1, r_2, \ldots, r_t)$. Then, the first two moments of the return distribution at time t can be written as $E[r_t|R_{t-1}] = \mu_t$ and

$$Var[r_t|R_{t-1}] = E[(r_t - \mu_t)^2|R_{t-1}] = E[e_t^2|R_{t-1}] = \sigma_t^2.$$

Besides the constant a_0, free parameters in the model are the mean $\alpha_0 > 0$, the persistences β_j and the moving average coefficients α_i. These parameters are restricted to insure positive variance ($\alpha_i \ge 0$, $\beta_j \ge 0$) and stationarity ($\sum_{i=1}^{q} \alpha_i + \sum_{j=1}^{p} \beta_j < 1$).

The GARCH model given by Eqs. (1)–(3) has the capacity to capture the main features of typical return series, namely excess kurtosis, small autocorrelation and high persistence of squared returns [9]. However, linear GARCH models often fail to capture all these features simultaneously. One approach to address this problem that can specifically help to account for the excess kurtosis and skewness is through treatment by non-Gaussian distributions.

2.2. RMDN-GARCH(p,q) network model

We construct a time-conditional mixture model for returns on asset prices using density functions $\phi_i(\mu_{i,t}, \sigma_{i,t}^2)$ whose arguments $\mu_{i,t} \equiv \mu_{i,t}(r_{t-1})$ and $\sigma_{i,t} \equiv \sigma_{i,t}^2(r_{t-1})$ are defined by nonlinear functions as follows:

$$p(r_t|R_{t-1}) = \sum_{i=1}^{N} \eta_{i,t} \phi_i(\mu_{i,t}, \sigma_{i,t}^2) \tag{4}$$

where $\eta_{i,t} \equiv \eta_{i,t}(r_{t-1})$ are the mixing coefficients that change with the time, ϕ_i represents the conditional density function of the returns r_t for the ith mixture component, and i is an index running over the N mixands. The coefficients should satisfy $\sum_{i=1}^{N} \eta_{i,t} = 1$ ($0 < \eta_{i,t} < 1$).

This density function ϕ_i is typically chosen to be Gaussian:

$$\phi_i(\mu_{i,t}, \sigma_{i,t}^2) = \frac{1}{\sqrt{2\pi\sigma_{i,t}^2}} \exp\left(-\frac{(r_t - \mu_{i,t})^2}{2\sigma_{i,t}^2}\right) \tag{5}$$

where dependencies on the past inputs is omitted for brevity.

It has been shown that such a mixture model is a universal approximator in the sense that it can approximate any conditional distribution $r_t|R_{t-1} \sim \mathcal{D}$ with mean μ_t and variance σ_t^2 with an arbitrary accuracy, provided that enough mixture components are available and the parameters are appropriately selected [26].

The moments of the mixture distribution $p(r_t|R_{t-1})$ (Eq. (4)) can be computed as follows:

$$\mu_t = \sum_{i=1}^{N} \eta_{i,t}\, \mu_{i,t} \tag{6}$$

$$\sigma_t^2 = \sum_{i=1}^{N} \eta_{i,t} (\sigma_{i,t}^2 - (\mu_{i,t} - \mu_t)^2). \tag{7}$$

Such a mixture model is implemented here as a hybrid RMDN-GARCH neural network architecture which produces non-linearities through three modules: (1) a mixing network that infers the mixing

coefficients $\eta_{i,t}$[1]; (2) a mean level network that generates the means $\mu_{i,t}$; and (3) a variance recurrent network that produces the volatilities $\sigma^2_{i,t}$ (Fig. 1). All network modules are designed to have one hidden node with a linear function to feed one output node (so as to generate exactly the linear part of the corresponding model), while the remaining hidden nodes are made to use the hyperbolic tangent activation function to produce non-linear extensions.

The mixing and mean level networks are feedforward multilayer perceptrons that yield non-linear autoregressive models using the same inputs (return r_{t-l} and constant 1) but through different output activations:

$$\eta_{i,t} = s\left(\sum_{k=1}^{K} u_{ik} g\left(\sum_{l=1}^{D} U_{kl} r_{t-l} + U_{k0}\right) + u_{i0}\right) \tag{8}$$

$$\mu_{i,t} = \sum_{k=1}^{K} v_{ik} g\left(\sum_{l=1}^{D} V_{kl} r_{t-l} + V_{k0}\right) + v_{i0} \tag{9}$$

where s is the softmax activation function $s(y_i) = \exp(y_i)/\sum_{j=1}^{N}\exp(y_j)$ [23], g is the hyperbolic tangent function $g(y) = \tanh(y)$, D is the number of lagged inputs, K is the number of hidden nodes, U_{kl} (V_{kl}) is the weight from the lth input node to the kth hidden node, the weight u_{ik} (v_{ik}) connects the kth hidden node to the ith output, U_{k0} (V_{k0}) and u_{i0} (v_{i0}) are constants.

The recurrent variance network consists of a layer of hidden nodes, and a layer of output nodes that compute volatilities $h_{i,t} = \sigma^2_{i,t}$. The linear hidden nodes in this network generate exactly the linear part of the model Eq. (3). The output nodes use the absolute value function to guarantee positiveness of the volatility [33]. The variance network accepts the errors $e^2_{t-l} = (y_{t-l} - \mu_{t-l})^2$ as external inputs and the volatilities σ^2_{t-l+p} as internal signals from its context.[2] The node activations serve as context that memorizes the past information. Thus, the output at a particular time step t depends not only on the recent returns, but also on the previous volatilities stored in its context.

Let us adopt the following notation for the inputs to the network at time t:

$$x_{t-l} = \begin{cases} 1 & \text{i.e. bias if } l = 0 \\ e^2_{t-l} & \text{if } 1 \le l \le p \\ h_{t-l+p} & \text{if } (p+1) \le l \le (p+q) \end{cases} \tag{10}$$

where p is the number of lagged inputs, and q is the number of recurrent connections.

These inputs feed the hidden nodes in the variance module to compute the summations

$$y_{k,t} = \sum_{l=0}^{p+q} W_{kl} x_{t-l} = \sum_{l=1}^{p} W_{kl} e^2_{t-l} + \sum_{l=p+1}^{p+q} W_{kl}\, h_{t-l+p} + W_{k0} \tag{11}$$

where W_{kl} are the input-to-hidden weights in the variance network, and l enumerates the inputs.

The hidden nodes pass these summations through the corresponding activation functions

$$z_{k,t} = g(y_{k,t}), \quad \text{and} \quad g(y_{k,t}) = \begin{cases} y_{k,t} & \text{linear} \\ \tanh(y_{k,t}) & \text{hyperbolic tangent} \end{cases} \tag{12}$$

where the activation functions can also be sigmoidal, as standard in neural network research.

The output nodes are made using the absolute value function $f(o_{n,t}) = |o_{n,t}|$ so as to generate outputs $h_{n,t} = \sigma^2_{n,t}$

$$\begin{aligned} h_{n,t} &= f(o_{n,t}) = f\left(\sum_{k=1}^{K} w_{nk} z_{k,t} + w_{n0}\right) \\ &= f\left(\sum_{k=1}^{K} w_{nk}\, g\left(\sum_{l=1}^{p} W_{kl} e^2_{t-l} + \sum_{l=p+1}^{p+q} W_{kl}\, h_{n,t-l+p} + W_{k0}\right) + w_{n0}\right) \end{aligned} \tag{13}$$

where n is the output node index, and W_{k0} and w_{n0} are the weights on connections feeding 1.

These dynamical equations (11)–(13) describe the nonlinear network model as sensitive not only to the external input variables r^2_{t-l} through the error e^2_{t-l}, but also to the temporal variables h_{t-l} at specific time steps. These temporal variables capture information from the past and send it via the recurrent connections, thus providing memory capacity. This is what helps to capture time-varying patterns in the data.

3. Dynamic learning of RMDN-GARCH

The learning problem is: given a trajectory of returns as training data $r_1, r_2, \ldots, r_{t,\ldots}$, sampled at discrete times $1 \le t \le T$, find the RMDN-GARCH parameters that best explain their changing variance. The learning task is to find weights which maximize the likelihood that the returns have been generated by a process which variance is the same as the one modeled by the network. Solutions are obtained through maximization of the instantaneous log-likelihood

$$L_t = \log\, p(r_t | R_{t-1}) = \log \sum_{i=1}^{N} \eta_{i,t} \phi_i(\mu_{i,t}, \sigma^2_{i,t}) \tag{14}$$

The total likelihood $L = \sum_{t=1}^{T} L_t$ as a function of the parameters in the recurrent mixture density network $\theta = \{(u_{ik})_{k=1}^{K}, (U_{kl})_{l=0}^{D}, (v_{ik})_{k=1}^{K}, (V_{kl})_{l=0}^{D}, (w_{ik})_{k=1}^{K}, (W_{kl})_{l=0}^{p+q}\}_{i=1}^{2}$ is minimized iteratively

$$\hat{\theta}_{(i+1)} = \arg\max_{\theta}\left\{\sum_{t=1}^{T} L_t(\theta, \hat{\theta}_{(i)})\right\} \tag{15}$$

starting with the estimates $\theta_{(i)}$ from the previous ith step.

The Broyden–Fletcher–Goldfarb–Shanno (BFGS) [17] optimization method is preferred for training the RMDN-GARCH network. The rationale is that BFGS is a second-order (quasi-Newton) method that performs more efficient learning (stable convergence with reasonable complexity) compared to gradient backpropagation methods [3].

3.1. Likelihood derivatives

The BFGS algorithm requires to find the derivatives of the log-likelihood with respect to the weights in each particular network module. The differentiation of the log-likelihood leads to terms including the time-varying volatility, which help to capture the correlation in time series. We obtain the following analytical derivatives for the hidden-to-output node weights in the RMDN-GARCH network (see Appendix):

$$\frac{\partial L_t}{\partial u_{ik,t}} = \frac{\partial L_t}{\partial \pi_{i,t}} \frac{\partial \pi_{i,t}}{\partial u_{ik,t}} = (\kappa_{i,t} - \eta_{i,t}) \frac{\partial \pi_{i,t}}{\partial u_{ik,t}} \tag{16}$$

$$\frac{\partial L_t}{\partial v_{ik,t}} = \frac{\partial L_t}{\partial \mu_{i,t}} \frac{\partial \mu_{i,t}}{\partial v_{ik,t}} = \left(\kappa_{i,t} \frac{(r_t - \mu_{i,t})}{h_{i,t}}\right) \frac{\partial \mu_{i,t}}{\partial v_{ik,t}} \tag{17}$$

$$\frac{\partial L_t}{\partial w_{ik,t}} = \frac{\partial L_t}{\partial h_{i,t}} \frac{\partial h_{i,t}}{\partial w_{ik,t}} = \left(\frac{1}{2} \frac{\kappa_{i,t}}{h_{i,t}} \left(\frac{(r_t - \mu_{i,t})^2}{h_{i,t}} - 1\right)\right) \frac{\partial h_{i,t}}{\partial w_{ik,t}} \tag{18}$$

[1] The mixing network actually computes the prior probability $\eta_{i,t}$ that the ith mixture can infer the desired output r_t.

[2] The number of past values sent as inputs to this network is sufficient since we use a recurrent architecture.

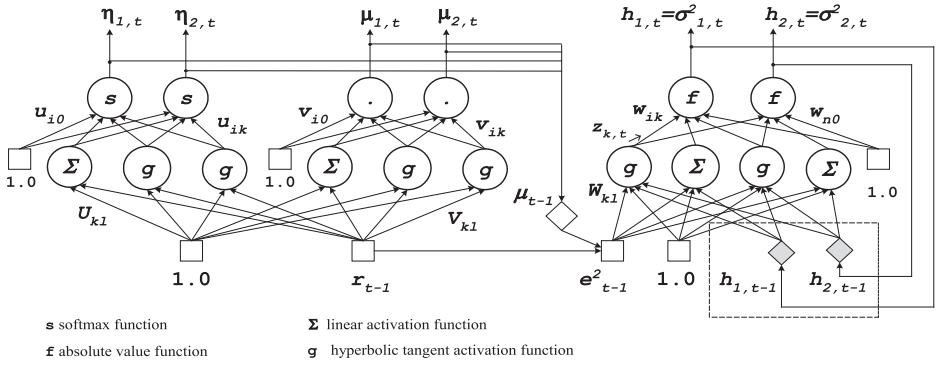

Fig. 1. RMDN-GARCH: a recurrent mixture density neural network interpretation of a nonlinear GARCH model. The outputs of the particular network modules generate the parameters of the conditional density model when given returns at their inputs (using also past volatilities from the context).

where $\pi_{i,t}$ is the weighted sum at the mixing network output $\pi_{i,t}=\sum_{k=1}^{K}u_{ik}g(U_{kl}r_{t-l}+U_{k0})+u_0$ (before passing it through the softmax activation), and $\kappa_{i,t}$ is the substitution[3] (see (5))

$$\kappa_{i,t}=\frac{\eta_{i,t}\ \phi_i}{\sum_{j=1}^{N}\eta_{j,t}\phi_j}.$$

The input-to-hidden node derivatives $\partial L_t/\partial U_{kl,t}$, $\partial L_t/\partial V_{kl,t}$, and $\partial L_t/\partial W_{kl,t}$ are obtained analogously.

The derivatives $\partial\pi_{i,t}/\partial u_{ik,t}$ and $\partial\pi_{i,t}/\partial U_{kl,t}$ in the mixing network, as well as $\partial\mu_{i,t}/\partial v_{ik,t}$ and $\partial\mu_{i,t}/\partial V_{kl,t}$ in the mean level network can be computed using the standard backpropagation algorithm for feedforward neural networks [31]. However, the log-likelihood derivatives $\partial h_{i,t}/\partial w_{ik,t}$ and $\partial h_{i,t}/\partial W_{kl,t}$ in the variance network are dynamic and need a special treatment.

3.2. Temporal derivatives

The temporal derivatives $\partial L_t/\partial w_{ik,t}$ and $\partial L_t/\partial W_{kl,t}$ in the variance network are computed according to the RTRL algorithm [37,38] for recurrent neural networks in order not only to adjust the weights using the instantaneous error information, but also to take advantage of the full error flow through time. RTRL calculates the temporal derivatives by evaluating recursively and storing partial derivatives during a forward pass through the network.

The derivative of the output $h_{n,t}$ with respect to a particular hidden-to-output weight w_{nk}, $1\le n\le N$, $1\le k\le K$, at time t is (see Appendix)

$$\frac{\partial h_{n,t}}{\partial w_{nk}}=f'(o_{n,t})z_{k,t} \tag{19}$$

where $f'(o_{n,t})$ is the derivative of the absolute value function f defined by $f'(o_{n,t})=o_{n,t}/|o_{n,t}|$.

The temporal derivative of the output $h_{n,t}$ with respect to an input-to-hidden node weight W_{ij}, $1\le i\le K$, $0\le j\le(p+q)$, at time t is computed in two steps (see Appendix)

$$\frac{\partial h_{n,t}}{\partial W_{ij}}=f'(o_{n,t})\left(\sum_{k=1}^{K}\left[w_{nk}\frac{\partial z_{k,t}}{\partial W_{ij}}\right]+\delta_{ni}z_{j,t}\right) \tag{20}$$

$$\frac{\partial z_{k,t}}{\partial W_{ij}}=g'(y_{k,t})\left(\sum_{l=p+1}^{p+q}\left[W_{kl}\frac{\partial h_{t-l+p}}{\partial W_{ij}}\right]+\delta_{ik}x_{t-j}\right) \tag{21}$$

where δ_{ni} is the Kroneker delta function: $\delta_{ni}=1$ if $n=i$ and 0 otherwise, the initial state of the network is considered independent from the weights, that is $\partial h_0/\partial W_{ij}=0$, and $g'(y_{k,t})$ denotes the derivative of g which for tangential activations is $g'(y_{k,t})=(1-y_{k,t}^2)$.

The time argument of the weights is omitted in Eqs. (20) and (21) assuming that they are modified slightly at each temporal training step. Under the assumption $\partial h_{n,t-l+p}/\partial W_{ij,t}\approx\partial h_{n,t-l+p}/\partial W_{ij,t-1}$ we can compute the quantity $\partial h_{n,t-l+p}/\partial W_{ij,t-1}$ and use it to obtain the weight updates. Having this simplification, however, means that only an approximation to the exact gradient is computed [38]. Practically the approximation is almost identical to the precise gradient when the learning rate is sufficiently small.

The time complexity of this RTRL training algorithm is $\mathcal{O}(q^3(q+p))$, and its memory requirements are proportional to $\mathcal{O}(q^2(q+p))$ [38]. The required memory is constant and does not change with the increase of the number of the data points. Although the complexity of this RTRL algorithm for training the recurrent neural network module is high, it is reasonable to apply it to small networks like the constructed here recurrent variance network because we typically use small dimensions $q=1$ (or 2) and $p=2$.

4. Applications to volatility inference

Experiments in volatility inference were conducted using artificially generated series, and a benchmark series of DEM/GBP exchange rates. Daily exchange rates were taken and transformed into percentage nominal returns. The research was carried out to examine: (1) the effect of separating the linear from the nonlinear parts in the mixture network density model; (2) the influence of dynamic training on the performance of recurrent non-linear mixture density network models; and (3) to relate them to standard linear GARCH models, as well as to models using Student-t error distributions.

4.1. Processing simulated series

The logistic mixture model: The ability of the proposed here approach to learn descriptions of stochastic processes was tested over series generated by a logistic mixture model [39]. The model was made flexible so that not only the volatility is time dependent but also the mixing proportions are changing over time as follows:

$$\begin{aligned}
&p(r_t|R_{t-1})=\eta_{1,t}\phi(\mu_t,\sigma_{1,t}^2)+\eta_{2,t}\phi(\mu_t,\sigma_{2,t}^2)\\
&\mu_t=a_0+a_1r_{t-1}\\
&\sigma_{1,t}^2=\alpha_{01}+\alpha_{11}e_{t-1}^2+\beta_1\sigma_{1,t-1}^2\\
&\sigma_{2,t}^2=\alpha_{02}+\alpha_{12}e_{t-1}^2+\beta_2\sigma_{1,t-1}^2\\
&\pi_{1,t}=c_0+c_1r_{t-1}\\
&\eta_{1,t}=\exp(\pi_{1,t})/(1+\exp(\pi_{1,t}))
\end{aligned} \tag{22}$$

[3] The variables $\kappa_{i,t}$ are posterior responsibilities which are functions of the inputs as well as the outputs of the mixtures, that is $\kappa_{i,t}$ is the posterior probability that the t-th mixture can generate the desired return r_t.

where ϕ is the Gaussian function, Eq. (5), the other mixing coefficients are computed by $\eta_{2,t}=(1-\eta_{1,t})$, the constants in the mean equation are $a_0=0.01$, $a1=0.4$, the volatility regime parameters are: $a_{01}=0.01$, $\alpha_{11}=0.1$, $\beta_1=0.75$, $a_{02}=0.04$, $\alpha_{12}=0.15$, $\beta_2=0.8$, and the constants in the equation for the mixing coefficient are $c_0=0.01$, $c_1=0.95$. The initial values are: $r_0=0.1$, $e_0^2=0$ and $\sigma_0^2=0$.

Using this process model there were generated three groups of series with increasing length as follows: $T=500$, $T=1000$, and $T=1500$. From each length we made 1000 replicas using sampling from the bimodal distribution defined by Eq. (22).

Network architecture and training: Since the objective of this simulation study was to evaluate the learning potential of RMDN-GARCH and to compare it with the previous RMDN, we designed the network as in the previous research [32,33] with three hidden nodes in each network. Each network module had two inputs (constant 1 and r_{t-1}), three hidden and two output nodes (Fig. 1). This is actually an RMDN(2,3,2)-GARCH(1,1) model whose volatility equation has one persistence β and one moving average coefficient α. The initial weights on links feeding the linear terms in the mixing network were computed using iteratively reweighted least squares (IRLS) as suggested for such architectures using softmax activations [23], and all remaining weights were set to zero. The initial weights on connections feeding the linear terms in the mean level net were computed using ordinary least squares (OLS) fitting an autoregressive AR(1) model, while the remaining weights were set to zero. The linear weights of the recurrent variance network were initialized with sensible values as follows: $a_{01}=0.005$, $\alpha_{11}=0.15$, $\beta_1=0.8$, $a_{02}=0.005$, $\alpha_{12}=0.2$, $\beta_2=0.85$.

Experimental results: Table 1 gives the estimated average parameters and their root mean squared errors (RMSE) in parentheses. The RMDN-GARCH mixture density network was trained with a BFGS optimizer [3] made with the proposed here temporal derivatives. The settings of the optimizer were: *Tolerance* $=1.0e^{-10}$, *MaxIterations* $=10^2$ and *FunctionEvaluations* $=10^2$. It should be noted that RMDN-GARCH is a nonlinear model with much more parameters, while we give in Table 1 only the parameters corresponding to the linear terms in the model defined by Eq. (22). The values in this table show that the dynamic optimization of RMDN-GARCH leads to adequate parameters, although the precision varies with the sample sizes.

Table 1
Estimated average RMDN-GARCH parameters and their RMSE errors in parentheses obtained after independent runs over 1000 simulated series of increasing sizes T (500,1000,1500) generated with the chosen true parameters for the logistic mixture model (Eq. (22)).

Parameter	True	*RMDN-GARCH*		
		$T=500$	$T=1000$	$T=1500$
a_0	0.01	0.0113 (0.0195)	0.0112 (0.0176)	0.0108 (0.0141)
a_1	0.40	0.3921 (0.0529)	0.3958 (0.0422)	0.3967 (0.0396)
α_{01}	0.01	0.0083 (0.0226)	0.0098 (0.0182)	0.0097 (0.0095)
α_{11}	0.10	0.1033 (0.0310)	0.0977 (0.0282)	0.0964 (0.0253)
β_1	0.75	0.7621 (0.0298)	0.7623 (0.0241)	0.7625 (0.0209)
α_{02}	0.04	0.0375 (0.0154)	0.0332 (0.0138)	0.0346 (0.0122)
α_{12}	0.15	0.1657 (0.0302)	0.1633 (0.0258)	0.1625 (0.0224)
β_2	0.80	0.8076 (0.0285)	0.8069 (0.0241)	0.8061 (0.0203)
c_0	0.01	0.0231 (0.0484)	0.0254 (0.0441)	0.0243 (0.0362)
c_1	0.95	0.9408 (0.1065)	0.9541 (0.0892)	0.9575 (0.0714)

Table 2
Average errors of fitting the high and low volatility regimes, computed by training the two different network architectures RMDN-GARCH and RMDN-OLD using the same BFGS optimizer with the same settings each over the same 1000 simulated series.

Algorithms	Regime	Error			
		RMSE	*MAE*	*RAE*	*MPE*
RMDN-GARCH $T=500$	1	0.0664	0.0428	0.9294	8.1975
	2	0.1058	0.0754	0.9854	5.5863
RMDN-OLD $T=500$	1	0.0783	0.0551	0.9733	11.2245
	2	0.1294	0.0822	0.9982	8.1954
RMDN-GARCH $T=1000$	1	0.0568	0.0348	0.7265	6.2217
	2	0.0906	0.0641	0.9143	4.3528
RMDN-OLD $T=1000$	1	0.0682	0.0514	0.8816	10.3519
	2	0.1178	0.0793	0.9624	7.8711
RMDN-GARCH $T=1500$	1	0.0487	0.0291	0.6027	5.3349
	2	0.0826	0.0547	0.8218	3.8254
RMDN-OLD $T=1500$	1	0.0605	0.0417	0.8295	9.5972
	2	0.1001	0.0661	0.9158	5.1682

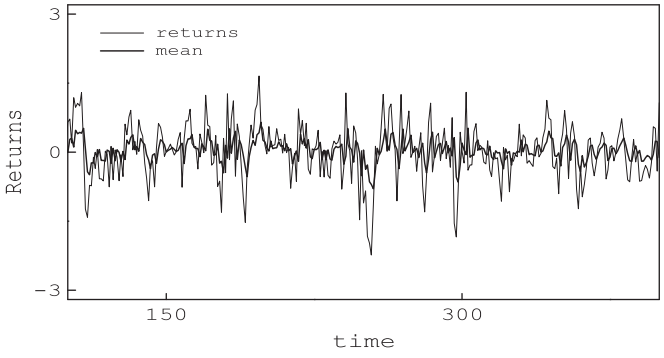

Fig. 2. A generated return series using the logistic mixture model and its approximation by the mean network module (as a mixture of the two outputs) produced after training the RMDN(2,3,2)-GARCH(1,1) network model.

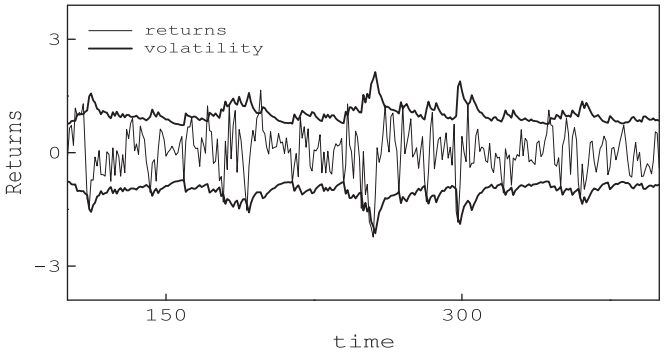

Fig. 3. Generated return series using the logistic mixture model and its time-varying volatility (conditional variance) inferred by the recurrent network module after training the RMDN(2,3,2)-GARCH(1,1) model.

Runs over the same simulated time series were also conducted with the previous network RMDN-OLD [33]. Both RMDN-GARCH and RMDN-OLD were deliberately trained with the same optimizer using the dynamic temporal derivatives proposed here to make clear the differences in their performance due only to the differences in their architectures. Table 2 offers averaged fitting errors from the two different network architectures RMDN-GARCH and RMDN-OLD each over the same 1000 simulated series. These fitting errors were computed as the mean errors of approximating the two true volatility regimes $\sigma_{1,t}^2$ and $\sigma_{2,t}^2$, $1\le t\le T$, using the

corresponding equations from Eq. (22). The results in Table 2 indicate that the separation of the linear from the nonlinear parts in the model leads to more accurate models with lower root mean squared error (RMSE), lower mean absolute error (MAE), lower relative average error (RAE) (also known as the Theil's U2 statistic) and lower mean percentage error (MPE). These results provide empirical evidence that the developed RMDN-GARCH is an improvement over the previous mixture density network architecture.

Evidence for the quality of fitting the moments of the conditional distribution of one of the generated series is provided in Figs. 2–4. Fig. 2 plots a return series from a particular run along with the computed mean return $\mu_t = \sum_{i=1}^{2} \eta_{i,t}\mu_{i,t}$ by the mean level module of the RMDN-GARCH network. Figs. 3 and 4 illustrate the inferred volatility obtained in the same run. The curves in Fig. 3 demonstrate that the inferred volatility wraps closely the deviations of the given returns, which indicates that the learning algorithm works properly. Fig. 4 gives the approximated low and high volatility regimes by the two outputs of the variance network module of RMDN-GARCH. One can see that the variance network identifies accurately the two regimes as its outputs are very close to the given simulated regimes.

Fig. 5 offers a plot of the probability density function inferred by the RMDN-GARCH mixture model, obtained with the averaged parameters from all runs over the series of size 500, along with the initial density obtained with the starting parameter values, and the true density of the generative logistic mixture model. All posterior distributions are obtained using nonparametric density estimation. It can be observed in Fig. 5 that the learned posterior distribution is close indeed to the true, unknown posterior distribution.

Fig. 6 illustrates the evolution of the log-likelihood function averaged after runs using the optimizer to learn the RMDN-GARCH model over one series of size 500.

4.2. Processing real-world series

4.2.1. Studied models and algorithms

Linear model estimation: There were implemented two algorithms for processing standard linear GARCH models: a maximum likelihood estimation *MLE* algorithm using analytical derivatives [16], and a Markov Chain Monte Carlo *MCMC* sampling algorithm [24]. The *MLE* estimation was carried out with the same BFGS optimizer [3] in order to facilitate comparisons.

In all experiments the BFGS optimizer was executed with the following settings: $Tolerance = 1.0e^{-10}$, $MaxIterations = 10^2$ and $FunctionEvaluations = 10^2$. It should be noted that BFGS is a batch (offline) algorithm that operates on the entire series taken as a whole. Concerning these settings we would like to clarify that

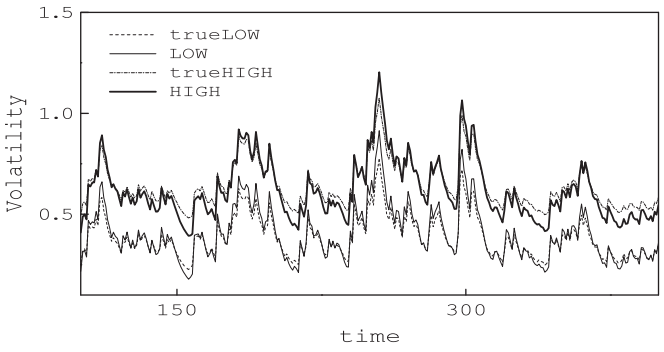

Fig. 4. High and low volatility regimes extracted as components of the common time-varying volatility from Fig. 1 obtained from the outputs of the RMDN(2,3,2)-GARCH(1,1) variance network module.

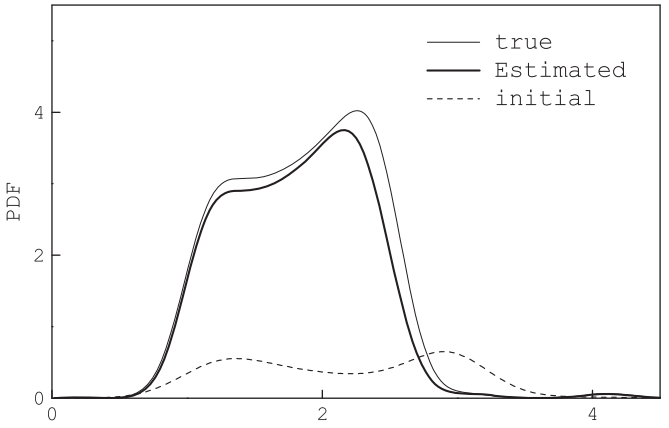

Fig. 5. Probability density function inferred by the RMDN(2,3,2)-GARCH(1,1) model, obtained with the averaged parameters from all runs over the series of size 500, along with the starting density obtained with the initial parameters, and the true density of the generative logistic mixture model.

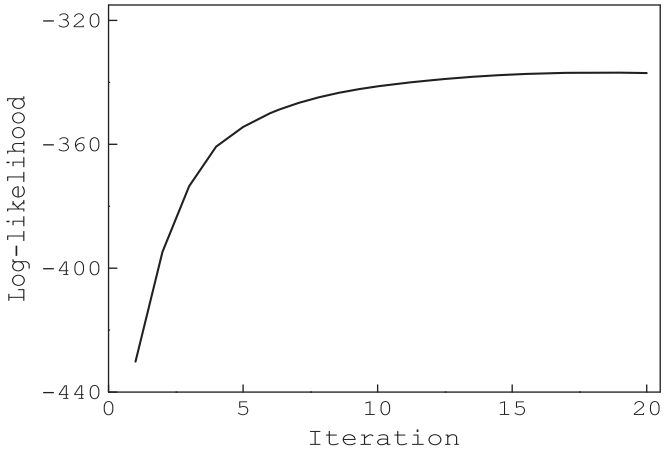

Fig. 6. Evolution of the log-likelihood function averaged after all the runs of the optimizer to estimate the RMDN(2,3,2)-GARCH(1,1) model over one series of size 500.

there were conducted a large number of preliminary experiments to determine these values so that the training process is fast and efficient, that is there are no substantial error decreases after doing more than the selected number of iterations and function evaluations with the chosen tolerance level.

The *MCMC* algorithm [24] uses adaptive rejection metropolis sampling. The sampler operates using priors and a likelihood function programmed for a linear GARCH. This *MCMC* algorithm was run for 5000 iterations after a burn-in period of 1000 iterations. All the algorithms were started with the same initial values: mean $\mu = 0.01$, persistence $\beta = 0.85$ and coefficient $\alpha = 0.05$. The volatility was initialized with the unconditional variance $\sigma_0^2 = \mu/(1-\alpha-\beta)$ [9].

Learning non-linear models: The proper architecture of the RMDN-GARCH(1,1) network model for this task was determined using a model selection technique, after dividing the data into a training subset of the first 80% points, and a testing subset with the remaining points. We started by estimating an RMDN network with two hidden nodes in each of the mixing, mean level, and variance networks, and computed the out-of-sample normalized mean squared error (*NMSE*) over the testing subset. Next, we expanded the hidden layer in each network to 3, 4, and 5 nodes and estimated the out-of-sample error of the corresponding model. The lowest *NMSE* was attained by the model with three

hidden nodes, so we selected the RMDN(2,3,2)-GARCH(1,1) topology for the following experiments.

The weights in the mean level network on links feeding the linear terms were computed using ordinary least squares (OLS) fitting of an autoregressive AR(1) model, while the remaining weights were set initially to zero. The weights in the mixing network on connections feeding the linear terms were initialized using iteratively reweighted least squares (IRLS) [23], and all remaining weights were set to zero. The weights in the linear part of the recurrent variance network were initialized with plausible values as follows: $a_{01}=0.005$, $\alpha_{11}=0.15$, $\beta_1=0.8$, $a_{02}=0.005$, $\alpha_{12}=0.2$, $\beta_2=0.85$.

The RMDN-GARCH was also trained with the BFGS optimizer [3], which was implemented using backprop derivatives for learning the mixing and mean level networks. Three different implementations of the optimizer were made for the recurrent variance network: $RMDN_{RTRL}$ using the temporal derivatives obtained in this paper, $RMDN_{BP}$ using static backprop derivatives as in previous research [32,33], and $RMDN_{ND}$ using static derivatives obtained by numerical differentiation. The objective was to find out whether the dynamic learning improves the results on the same network architecture.

Inference with heavy-tailed noise: We also implemented corresponding versions of our models using the Student-t likelihood function to investigate whether using this heavy tail distribution can help to achieve better results. The following algorithms were designed: $MCMC_t$, MLE_t, and a single dynamic recurrent density network with one-mixand RDN_t trained using RTRL derivatives. The degrees of freedom parameter ν of the Student-t distribution were found by $MCMC_t$ sampling to be $\nu=6.05$, and this value was then also used in MLE_t and RDN_t.

4.2.2. Experimental technology

Investigations were carried out to evaluate the impact of the volatility dynamics on the in-sample and out-of-sample performance of the models. The given series was split into training and testing subseries. The training series was used to infer the model parameters and their standard errors. The out-of-sample accuracy was examined by computing one-step-ahead volatility forecasts, and rolling sequentially by one-step foreword over the testing subseries. In particular, using information R_{t-1} up to time $t-1$, the volatility σ_t^2 at time t was calculated and compared to its observed proxy—the squared return r_t^2. After predicting the volatility the model was retrained, and this algorithm repeated till the end of the testing series. The training subseries were overlapping but their size was constant.

First, in-sample and out-of-sample statistical diagnostics were obtained using the standardized residuals $\hat{\varepsilon}_t=(r_t-\mu_t)/\sqrt{h_t}$. We calculated the coefficients of skewness and kurtosis, Durbin–Watson (*D–W*) and Ljung–Box (*L–B*) autocorrelation statistics, and the log-likelihood. Second, the predicted volatilities were taken to calculate several measures of out-of-sample performance, namely the normalized mean squared error (*NMSE*), the normalized mean absolute error (*NMAE*), the hit rate (*HR*) and the weighted hit rate (*WHR*) [33]

$$NMSE=\sqrt{\sum_{t=1}^{T}(r_t^2-\sigma_t^2)^2}\Big/\sqrt{\sum_{t=1}^{T}(r_t^2-r_{t-1}^2)r^2}$$

$$NMAE=\sum_{t=1}^{T}|r_t^2-\sigma_t^2|\Big/\sum_{t=1}^{T}|r_t^2-r_{t-1}^2|$$

$$HR=(1/T)\sum_{t=1}^{T}\gamma_t,$$

$$\text{where } \gamma_t=\begin{cases}1 & \text{if } (\sigma_t^2-r_{t-1}^2)(r_t^2-r_{t-1}^2)\geq 0\\ 0 & \text{otherwise}\end{cases} \tag{23}$$

Table 3
Computed linear and non-linear GARCH(1,1) model parameters with different estimation algorithms and their standard errors, obtained over the training series of 1500 daily returns on DEM/GBP currency exchange rates.

Algorithms	α_0	α_1	β
MLE	0.0121	0.1475	0.8064
	(0.0047)	(0.0395)	(0.0435)
MCMC	0.0126	0.1395	0.8107
	(0.0041)	(0.0342)	(0.0434)
$RMDN_{ND}$	0.0284	0.1954	0.7957
	(0.0044)	(0.0398)	(0.0552)
	−0.0051	0.1396	0.7624
	(0.0023)	(0.0462)	(0.0561)
$RMDN_{BP}$	0.0231	0.2086	0.7820
	(0.0038)	(0.0341)	(0.0421)
	−0.0039	0.1406	0.7343
	(0.0018)	(0.0357)	(0.0436)
$RMDN_{RTRL}$	0.0345	0.1863	0.7981
	(0.0032)	(0.0272)	(0.0435)
	−0.0087	0.0779	0.8653
	(0.0017)	(0.0308)	(0.0407)
RDN_t	0.0196	0.1875	0.8493
	(0.0041)	(0.0273)	(0.0382)
$MCMC_t$	0.0116	0.1320	0.8228
	(0.0035)	(0.0252)	(0.0377)
MLE_t	0.0114	0.1451	0.8127
	(0.0043)	(0.0285)	(0.0386)

$$WHR=\sum_{t=1}^{T}s_t|r_t^2-r_{t-1}^2|/\sum_{t=1}^{T}|r_t^2-r_{t-1}^2|,$$

$$\text{where } s_t=\text{sgn}((\sigma_t^2-r_{t-1}^2)(r_t^2-r_{t-1}^2)) \tag{24}$$

which relate the volatility estimate from the actual network model σ_t^2 to the target return r_t^2 assuming that it is the true volatility generated by a naive model.

The *NMAE* measure is less sensitive to outliers than the *NMSE*, although they both evaluate the discrepancy between the inferred volatility σ_t^2 by the actual model, and the true volatility r_t^2 from the naive benchmark model. The HR and WHR measure the frequency of correctly predicted directional changes by the models. The *HR* varies between 0 and 1, and a value of 0.5 means that the model is not better than a random generator. The *WHR* is in the worst case −1 and in the best 1.

The effectiveness of the models in out-of-sample forecasting was investigated using bootstrapped replicates of the DEM/GBP series. We followed a bootstrapping technology [29] for GARCH models. After fitting the model, its parameter estimates $\hat{a},\hat{b},\{\hat{\alpha}_0,\hat{\alpha}_1,\hat{\beta}\}$ were taken to compute standardized residuals $\hat{\varepsilon}_t=(r_t-\mu_t)/\sqrt{h_t}$. Next, samples from these standardized residuals were drawn $\hat{\varepsilon}_t^*=\hat{\varepsilon}_t-mean(\hat{\varepsilon}_t)$, in order to produce replicates of the returns $\{r_1^*,r_2^*,\ldots,r_T^*\}$ with the equation: $r_t^*=\mu_t^*+\hat{\varepsilon}_t^*\sqrt{h_t^*}, 1\leq t\leq T$. This includes recursive calculation of time-dependent volatilities h_t^* as well as calculation of means by $\mu_t^*=\hat{a}+\hat{b}r_{t-1}^*$ at each time step using the parameters $\{\hat{\alpha}_0,\hat{\alpha}_1,\hat{\beta}\}$. The process started with $\mu_1^*=\hat{\mu}_1$ and $h_1^*=\hat{h}_1$. After that, the parameters were re-estimated over the replicated returns leading to a new set of adapted parameters $\hat{a}^*,\hat{b}^*,\{\hat{\alpha}_0^*,\hat{\alpha}_1^*,\hat{\beta}^*\}$. Thus, we obtained 100 bootstrap replicates from the DEM/GBP series which were taken to study the predictive performance of the considered models.

4.2.3. Learning from the DEM/GBP series

A series of DEM/GBP currency exchange rates was taken [8] as a benchmark [2]. The series consists of 1974 daily observations recorded from 3 January 1984 to 31 December 1991, which were divided into 1500 data for training and 474 for testing. Table 3 shows the values of the main three GARCH parameters found by the studied algorithms. These values are very close to these

obtained by relevant research: $\alpha_0 = 0.0108$, $\alpha_1 = 0.1531$ and $\beta = 0.80597$ [2].

Tables 4 and 5 report the results from statistical tests on the in-sample and out-of-sample standardized residuals from the models. It shows that *MCMC*, $MCMC_t$, $RMDN_{RTRL}$ and RDN_t lead to models with close low skewness and kurtosis. All models have similar values of the Durbin–Watson statistics close to 2.0, indicating that there is no significant autocorrelation in the residuals, which is confirmed by the Ljung–Box test. The nonlinear mixture model trained by $RMDN_{RTRL}$ outperforms the linear ones using Gaussian and Student-*t* noise with respect to the log-likelihood and several statistical characteristics (the *MCMC* algorithm also helps to better capture the kurtosis), while the $RMDN_{BP}$ and $RMDN_{ND}$ are inferior in this context to $RMDN_{RTRL}$ and RDN_t obtained using dynamic derivatives. Both algorithms $RMDN_{RTRL}$ and RDN_t learn models with slightly better statistical characteristics than the other volatility models. These nonlinear dynamic mixture density networks fit better the returns. Overall, all mixture models are more likely than the GARCH models as they show lower *GMLE* estimates.

4.2.4. Predictive performance

The average forecasting results are given in Table 6. One can see that the dynamically trained mixture $RMDN_{RTRL}$ network demonstrates improved performance, compared to the previous approaches. Our findings can be summarized as follows:

1. The lowest average *NMSE* and *NMAE* errors were obtained after dynamic training of the mixture $RMDN_{RTRL}$ model and the RDN_t model with *t*-Student noise—they beat not only the linear models, but also the static nonlinear mixture $RMDN_{BP}$ and $RMDN_{ND}$ models.
2. The dynamic training of the nonlinear network-based models $RMDN_{RTRL}$ and RDN_t lead to better average 'economic' performance as indicated by their higher *HR* and *WHR* rates. The linear model estimated by *MLE* achieved highest *HR*, while the linear model trained by Monte Carlo sampling using Student-*t* noise ($MCMC_t$) also showed good economic performance.
3. The highest average log-likelihood was achieved by the $RMDN_{RTRL}$ model, with MLE_t being the second, and RDN_t being the third.

The reasonable question that arises after looking at the results in Table 6 is as to what degree the differences between the models are statistically significant. In order to find out, we applied the paired *t*-test to the predicted average *NMSE* obtained from the studied models. There were considered the bootstrapped replicas of the original series, and there were produced 100 average *NMSE* errors from each model (that is, each model was rolled over the corresponding 100 replicated testing subseries, and the computed one-step ahead forecasting errors over each subseries path of size 474 were averaged). The key idea behind this technology was to generate independent error samples from each model, having also in mind that the errors are assumed to be normally distributed.

Tables 7 and 8 present the calculated *p*-values from the paired difference tests. Table 7 relates the $RMDN_{RTRL}$ model to the linear models, and Table 8 relates it to the nonlinear models trained with the corresponding algorithms. One can see in these tables that the null hypothesis that there is no difference between them at the 0.1 (0.9 percept) level is rejected. That is, the mean differences between the errors of the developed $RMDN_{RTRL}$ model and the remaining models are greater than zero.

Fig. 7a plots the returns and two curves produced by adding twice the square root of the inferred volatility. This figure shows that the deviations of the volatility learned by $RMDN_{RTRL}$ training of RMDN-GARCH wrap closely the given target returns. Fig. 7b presents a correlogram of the standardized residuals. The correlogram of the residuals demonstrates that the autocorrelation in the residuals up to 30 lags is small and close to zero. Fig. 7c gives a *Q*–*Q*

Table 4
In-sample statistical diagnostics calculated with standardized residuals, obtained by fitting several GARCH(1,1) to the training series of daily returns on DEM/GBP using the studied algorithms.

Algorithms	Skewness	Kurtosis	D–W	L–B(30)	Log-lik.
MLE	−0.3865	5.2274	1.9324	28.3489	−912.841
MCMC	−0.4006	5.1403	1.9160	29.2031	−913.447
$RMDN_{ND}$	−0.4053	5.3312	1.9115	29.7462	−865.221
$RMDN_{BP}$	−0.4026	5.2861	1.8961	29.1403	−862.682
$RMDN_{RTRL}$	−0.3971	5.1706	1.9117	30.3248	−838.525
RDN_t	−0.3685	5.3115	1.9053	28.5802	−857.724
$MCMC_t$	−0.3988	5.1332	1.9165	29.2136	−868.791
MLE_t	−0.3892	5.3718	1.9573	26.4898	−850.624

Table 5
Out-of-sample statistical diagnostics calculated with standardized residuals, obtained by fitting GARCH(1,1) to the training series of daily returns on DEM/GBP using the studied algorithms.

Algorithms	Skewness	Kurtosis	D–W	L–B(30)	Log-lik.
MLE	−0.4002	5.1838	1.9232	27.8543	−911.77
MCMC	−0.4001	5.1517	1.9157	29.1331	−913.38
$RMDN_{ND}$	−0.4072	5.2915	1.9156	29.6538	−859.59
$RMDN_{BP}$	−0.4027	5.2464	1.8941	29.1402	−857.62
$RMDN_{RTRL}$	−0.3975	5.1514	1.9116	30.3248	−833.52
RDN_t	−0.3912	5.2618	1.9054	28.7643	−852.98
$MCMC_t$	−0.3989	5.1552	1.9175	29.1615	−863.89
MLE_t	−0.4011	5.2130	1.9218	29.8568	−851.11

Table 6
Averaged out-of-sample errors and log-likelihoods of different GARCH estimation methods computed with one-step-ahead forecasts, obtained via rolling regression over the testing bootstrapped subseries of 474 future DEM/GBP exchange rates data.

Algorithms	NMSE	NMAE	HR	WHR	Log-lik.
MLE	0.7582	0.9158	0.6575	0.5768	−389.22
MCMC	0.7596	0.9317	0.6492	0.5779	−390.41
$RMDN_{ND}$	0.7726	0.9653	0.6473	0.5781	−367.34
$RMDN_{BP}$	0.7731	0.9641	0.6491	0.5627	−366.58
$RMDN_{RTRL}$	0.7562	0.9049	0.6535	0.5824	−356.76
RDN_t	0.7592	0.8982	0.6526	0.5795	−364.59
$MCMC_t$	0.7598	0.9246	0.6515	0.5792	−369.38
MLE_t	0.7661	0.9325	0.6497	0.5744	−363.15

Table 7
Computed *p*-values of the paired *t*-test (relating $RMDN_{RTRL}$ to the linear models) with null hypothesis that the averaged forecasting NMSE errors are not statistically different.

pair	*MLE*	*MCMC*	$MCMC_t$	MLE_t
$RMDN_{RTRL}$	0.1561	0.1424	0.1587	0.1611

Table 8
Computed *p*-values of the paired *t*-test (relating $RMDN_{RTRL}$ to the nonlinear models) with null hypothesis that the forecasting NMSE errors are not statistically different.

pair	$RMDN_{ND}$	$RMDN_{BP}$	RDN_t
$RMDN_{RTRL}$	0.1557	0.1125	0.1229

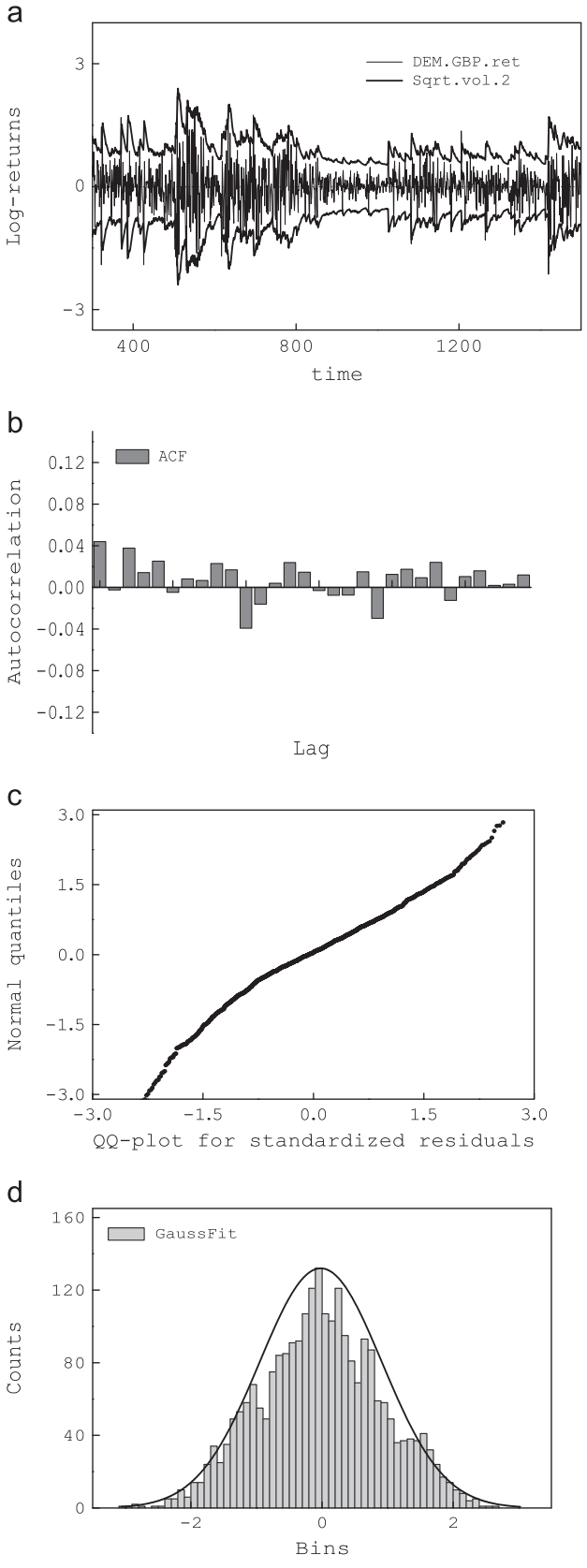

Fig. 7. (a) Return series from DEM/GBP exchange rates, recorded from 3 January 1984 to 31 December 1991, and their time-varying volatility (conditional variance) curves produced by an RMDN-GARCH(1,1) model estimated by BFGS optimization using RTRL temporal derivatives. (b) Correlogram of the standardized residuals computed with RMDN-GARCH(1,1) volatilities. (c) Quantile–quantile plot of the standardized residuals computed with RMDN-GARCH(1,1) volatilities. (d) Histogram of the standardized residuals computed with RNN-GARCH(1,1) volatilities.

plot that relates the standardized residuals to the normal distribution, and illustrates that the residuals are almost normal in the middle and only deviate from normality in the tails at the corners, i.e. there is no significant deviation from normality in them. Fig. 7d offers a Gaussian approximation to the histogram of the standardized residuals.

Fig. 8 illustrates the confidence intervals of the forecasted volatility to show its degree of variability over the bootstrapped replicas of the financial exchange rates series.

4.2.5. *Discussion*

The main objective of our approach is modeling and prediction of the (unobserved) instantaneous second-order moment of the time-dependent distributions of time series. This is a micro-level view of the underlying volatility process, which is different from the macro-level view of complexity science [12,35]. In particular, we model in detail the heteroscedastic nature of the stochastic process given the observed data. As in any modeling approach, we impose certain assumptions on the nature of state-conditional noise and dependency structure over time. The appropriateness of our assumptions is then tested on a hold-out sample of data not used in model building.

This research uses mixtures of Gaussians to approximate any 'reasonable' (e.g. smooth) density function to arbitrary precision assuming a sufficient number of mixture components. However, the use of Gaussian mixtures in our approach has a stronger focus as financial returns are typically fat-tailed and a single Gaussian would be inappropriate. A two-component mixture allows us to describe a broad and a more peaked Gaussian, which is usually sufficient to capture the returns' variability. Alternatively, the Student *t*-distribution, with appropriately set number of degrees of freedom could be used, however, the mixture formulation is more general.

Modeling time-dependent variance and heteroscedastic noise have been addressed from various perspectives, for example within the popular kernel framework [10]. The main (not the only one) difference between this kernel approach and our approach is that while Cawley et al. formulate a 'static' regression model that can deal with different noise variances in different regions of the input space, our approach is dynamic and employs state-space modeling as the main representation mechanism for dealing with temporal dependencies. One can envisage using the kernel approach with a time lag (sliding input window). This however would require an imposition of a fixed model order. Our state-space approach is more flexible and general.

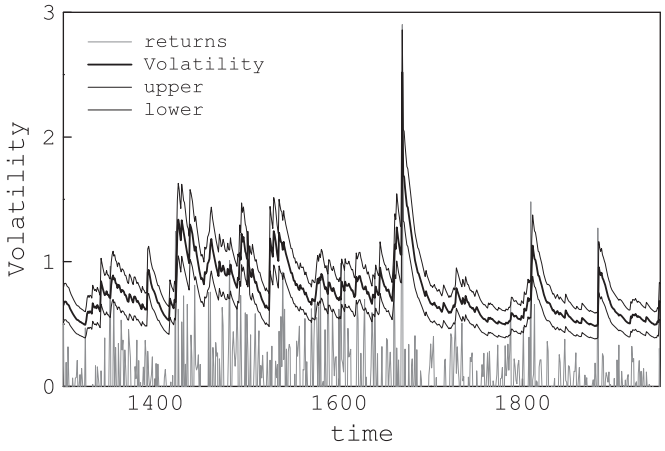

Fig. 8. Confidence intervals (95%) of time-varying volatility forecasts obtained by running the MT(2)-GARCH(1,1) model over 100 bootstrapped replicas of the DEM/GBP exchange rates series.

5. Conclusion

This paper presented a recurrent mixture density network approach to nonlinear volatility modeling of the dynamic evolution of the conditional variance in financial time series. The empirical results demonstrate that the presented approach learns accurate RMDN-GARCH models of the time-varying conditional moments of the returns, including the skewness and the kurtosis. It should be noted, however, that the 'vanishing gradient' problem cannot be easily avoided in any RMDN formulation (unless one considers specialized architectures). Nevertheless, the empirical investigations showed that temporal propagation of error information is useful in fitting GARCH models as RMDN-GARCH networks using temporal derivatives achieve good statistical characteristics and forecasting potential that outperform results from previous approaches. The model can be easily extended to include negative correlation between the return and the volatility. Various error functions that discount the effect of outliers can be applied to develop robust training algorithms for the mixture density network [1].

Further research will investigate as to what degree the RMDN-GARCH networks can produce accurate density forecasts and can be efficient in various practical financial applications, such as value-at-risk estimation and derivative pricing.

Appendix A. Gradient computations in the RMDN-GARCH network

The estimation of the mixture density recurrent RMDN-GARCH network weight parameters requires the computation of the gradients of the log-likelihood with respect to each of them. Each of the gradients, given by Eqs. (16)–(18), consist of two multipliers obtained following the chain rule. The first multiplier is the derivative of the instantaneous log-likelihood with respect to the particular network output, and the second is the output derivative with respect to the weight.

Likelihood derivatives in the mixing network: The log-likelihood derivatives in the mixing network are convenient to tackle with through the mixing network output sums $\pi_{i,t}=\sum_{k=1}^{K}u_{ik}g(U_{kl}r_{t-1}+U_{k0})+w_0$ according to Eq. (8), that is before passing them through the softmax activation function $s(\pi_{i,t})$ to compute the output $\eta_{i,t}=\exp(\pi_{i,t})/\sum_{j=1}^{N}\exp(\pi_{j,t})$. Then, the differentiation of the instantaneous log-likelihood L_t with respect to the mixing network output sums $\pi_{i,t}$ is carried out as follows [22]:

$$\frac{\partial L_t}{\partial \pi_{i,t}}=\sum_{n=1}^{N}\frac{\partial L_t}{\partial \eta_{n,t}}\frac{\partial \eta_{n,t}}{\partial \pi_{i,t}} \tag{A.1}$$

The first factor $\partial L_t/\partial\eta_{n,t}$ is obtained in the following way:

$$\frac{\partial L_t}{\partial \eta_{n,t}}=\frac{\phi_n}{\sum_{j=1}^{N}\eta_{j,t}\phi_j}\frac{\eta_{n,t}}{\eta_{n,t}}=\frac{\kappa_{n,t}}{\eta_{n,t}} \tag{A.2}$$

which uses for clarity the substitution

$$\kappa_{n,t}=\frac{\eta_{n,t}\phi_n}{\sum_{j=1}^{N}\eta_{j,t}\phi_j} \tag{A.3}$$

The calculation of the second factor $\partial\eta_{n,t}/\partial\pi_{i,t}$ involves two cases. First, when of $n=i$ the derivative is

$$\begin{aligned}\frac{\partial \eta_{n,t}}{\partial \pi_{i,t}}&=\frac{\exp(\pi_{n,t})\sum_{j=1}^{N}\exp(\pi_{j,t})-\exp(\pi_{n,t})\exp(\pi_{i,t})}{(\sum_{j=1}^{N}\exp(\pi_{j,t}))^2}\\&=\frac{\exp(\pi_{n,t})}{\sum_{j=1}^{N}\exp(\pi_{j,t})}-\frac{\exp(\pi_{n,t})\exp(\pi_{i,t})}{(\sum_{j=1}^{N}\exp(\pi_{j,t}))^2}\\&=\eta_{n,t}-\eta_{n,t}\eta_{i,t}\end{aligned} \tag{A.4}$$

Second, in case of $n\neq i$ the derivative is

$$\frac{\partial \eta_{n,t}}{\partial \pi_{i,t}}=\frac{-\exp(\pi_{n,t})\exp(\pi_{i,t})}{(\sum_{j=1}^{N}\exp(\pi_{j,t}))^2}=-\eta_{n,t}\eta_{i,t} \tag{A.5}$$

Considered together these two cases lead to the equation

$$\frac{\partial \eta_{n,t}}{\partial \pi_{i,t}}=\delta_{n,i}\eta_{n,t}-\eta_{n,t}\eta_{i,t} \tag{A.6}$$

using the Kroneker delta function $\delta_{n,i}=1$ if $n=i$ and 0 otherwise.

Therefore, the derivative of the instantaneous log-likelihood with respect to the output sums becomes

$$\begin{aligned}\frac{\partial L_t}{\partial \pi_{i,t}}&=\sum_{n=1}^{N}\frac{\kappa_{n,t}}{\eta_{n,t}}(\delta_{n,i}\eta_{n,t}-\eta_{n,t}\eta_{i,t})\\&=\frac{\kappa_{i,t}}{\eta_{i,t}}\eta_{i,t}-\sum_{n=1}^{N}\frac{\kappa_{n,t}}{\eta_{n,t}}\eta_{n,t}\eta_{i,t}\\&=\kappa_{i,t}-\eta_{i,t}\sum_{n=1}^{N}\kappa_{n,t}=\kappa_{i,t}-\eta_{i,t}\end{aligned} \tag{A.7}$$

where the variable $\kappa_{i,t}$ denotes a kind of posterior probability.

Likelihood derivatives in the mean level network: The differentiation of the instantaneous log-likelihood L_t with respect to the mean network outputs $\mu_{i,t}$ is carried out using the chain rule

$$\frac{\partial L_t}{\partial \mu_{i,t}}=\frac{\partial L_t}{\partial \phi_i}\frac{\partial \phi_i}{\partial \mu_{i,t}} \tag{A.8}$$

The first factor $\partial L_t/\partial\phi_i$ is obtained in the following way:

$$\frac{\partial L_t}{\partial \phi_i}=\frac{\pi_{i,t}}{\sum_{j=1}^{N}\pi_{j,t}\phi_j} \tag{A.9}$$

The second factor $\partial\phi_i/\partial\mu_{i,t}$ is calculated as follows:

$$\frac{\partial \phi_i}{\partial \mu_{i,t}}=\phi_i\frac{(-2(r_t-\mu_{i,t}))}{2h_{i,t}}(-1)=\phi_i\frac{(r_t-\mu_{i,t})}{h_{i,t}} \tag{A.10}$$

Hence, the derivative of the instantaneous log-likelihood with respect to the mean network outputs is

$$\frac{\partial L_t}{\partial \mu_{i,t}}=\kappa_{i,t}\frac{(r_t-\mu_{i,t})}{h_{i,t}} \tag{A.11}$$

where $\kappa_{i,t}$ is the variable already defined above.

Likelihood derivatives in the variance network: The differentiation of the instantaneous log-likelihood L_t with respect to the variance network outputs $h_{i,t}$ is carried out using the chain rule

$$\frac{\partial L_t}{\partial h_{i,t}}=\frac{\partial L_t}{\partial \phi_i}\frac{\partial \phi_i}{\partial h_{i,t}} \tag{A.12}$$

The first factor $\partial L_t/\partial\phi_i$ has already been computed above. The second factor $\partial\phi_i/\partial h_{i,t}$ is obtained in the following way:

$$\begin{aligned}\frac{\partial \phi_i}{\partial h_{i,t}}&=\frac{1}{\sqrt{2\pi}}\exp\left(-\frac{(r_t-\mu_{i,t})^2}{2h_{i,t}}\right)\left(-\frac{1}{2h_{i,t}\sqrt{h_{i,t}}}\right)\\&\quad+\frac{1}{\sqrt{2\pi h_{i,t}}}\exp\left(-\frac{(r_t-\mu_{i,t})^2}{2h_{i,t}}\right)\left[-\left(-\frac{(r_t-\mu_{i,t})^2}{2h_{i,t}}\right]\right)\\&=-\frac{1}{2}\phi_i\frac{1}{h_{i,t}}+\frac{1}{2}\phi_i\frac{(r_t-\mu_{i,t})^2}{h_{i,t}^2}\\&=-\frac{1}{2}\frac{\phi_i}{h_{i,t}}\left(1-\frac{(r_t-\mu_{i,t})^2}{h_{i,t}}\right)\\&=\frac{1}{2}\frac{\phi_i}{h_{i,t}}\left(\frac{(r_t-\mu_{i,t})^2}{h_{i,t}}-1\right)\end{aligned} \tag{A.13}$$

Finally, the derivative of the instantaneous log-likelihood with respect to the variance network outputs becomes

$$\frac{\partial L_t}{\partial h_{i,t}} = \frac{1}{2}\frac{\kappa_{i,t}}{h_{i,t}}\left(\frac{(r_t-\mu_{i,t})^2}{h_{i,t}}-1\right) \tag{A.14}$$

where $\kappa_{i,t}$ is an already defined variable.

Appendix B. Derivation of the RTRL algorithm for the recurrent variance network

Applying the RTRL algorithm [37,38] to compute the RMDN-GARCH training rules begins with computation of the derivative of the instantaneous loss with respect to each weight. According to the chain rule we obtain the following two kinds of derivatives for the hidden to output node weights w_{nk}, $1\le n\le N$, $1\le k\le K$, and for the input to hidden node weights W_{ij}, $1\le i\le K$, $0\le j\le(p+q)$:

$$\frac{\partial L_t}{\partial w_{nk}} = \frac{\partial L_t}{\partial h_{n,t}}\frac{\partial h_{n,t}}{\partial w_{nk}} = \varepsilon_t\frac{\partial h_{n,t}}{\partial w_{nk}} \tag{B.1}$$

$$\frac{\partial L_t}{\partial W_{ij}} = \frac{\partial L_t}{\partial h_{n,t}}\frac{\partial h_{n,t}}{\partial W_{ij}} = \varepsilon_t\frac{\partial h_{n,t}}{\partial W_{ij}} \tag{B.2}$$

where the derivatives $\varepsilon_t = \partial L_t/\partial h_{n,t}$ have been already obtained.

The derivatives of the output with respect to the incoming weights are taken in the following way:

$$\frac{\partial h_{n,t}}{\partial w_{nk}} = \frac{\partial h_{n,t}}{\partial o_{n,t}}\frac{\partial o_{n,t}}{\partial w_{nk}} = f'(o_{n,t})\frac{\partial\left[\sum_{k=1}^{K} w_{nk}z_{k,t}\right]}{\partial w_{nk}} = f'(o_{n,t})z_{k,t}$$

The derivatives of the output with respect to input to hidden weights are taken in two steps as follows:

$$\begin{aligned}\frac{\partial h_{n,t}}{\partial W_{ij}} &= \frac{\partial h_{n,t}}{\partial o_{n,t}}\frac{\partial o_{n,t}}{\partial W_{ij}}\\ &= f'(o_{n,t})\frac{\partial[\sum_{k=1}^{K} w_{nk}z_{k,t}]}{\partial W_{ij}}\\ &= f'(o_{n,t})\sum_{k=1}^{K}\left[w_{nk}\frac{\partial z_{k,t}}{\partial W_{ij}} + z_{k,t}\frac{\partial w_{nk}}{\partial W_{ij}}\right]\\ &= f'(o_{n,t})\left(\sum_{k=1}^{K}\left[w_{nk}\frac{\partial z_{k,t}}{\partial W_{ij}}\right] + \delta_{ni}z_{j,t}\right)\end{aligned} \tag{B.3}$$

$$\begin{aligned}\frac{\partial z_{k,t}}{\partial W_{ij}} &= \frac{\partial z_{k,t}}{\partial y_{k,t}}\frac{\partial y_{k,t}}{\partial W_{ij}} = g'(y_{k,t})\frac{\partial[[\sum_{l=0}^{p+q} W_{kl}x_{t-l}]}{\partial W_{ij}}\\ &= g'(y_{k,t})\sum_{l=0}^{p+q}\left[W_{kl}\frac{\partial x_{t-l}}{\partial W_{ij}} + x_{t-l}\frac{\partial W_{kl}}{\partial W_{ij}}\right]\\ &= g'(y_{k,t})\left(\sum_{l=1}^{p}\left[W_{kl}\frac{\partial e^2_{t-l}}{\partial W_{ij}}\right] + \sum_{l=p+1}^{p+q}\left[W_{kl}\frac{\partial h_{t-l+p}}{\partial W_{ij}}\right] + \delta_{ik}x_{t-j}\right)\\ &= g'(y_{k,t})\left(\sum_{l=p+1}^{p+q}\left[W_{kl}\frac{\partial h_{t-l+p}}{\partial W_{ij}}\right] + \delta_{ik}x_{t-j}\right)\end{aligned} \tag{B.4}$$

which uses the fact that $\partial e^2_{t-l}/\partial W_{ij} = 0$.

Both equations for the derivatives of the dynamic variables are similar in that their rightmost multipliers in the parentheses contain two similar terms. The first term accounts for the implicit effect of the weight W_{ij} on the network node activations $z_{k,t}$ and h_{t-l+p}, while the second term is the explicit effect of the weight W_{ij} on the particular ith network node.

References

[1] H. Allende, R. Torres, R. Salas, C. Moraga, Robust learning algorithm for the mixture of experts, in: Proceedings of the Iberian Conference on Pattern Recognition and Image Analysis, Lecture Notes in Computer Science, vol. 2652, Springer, Berlin, 2003, pp. 19–27.

[2] D. Ardia, Financial Risk Management with Bayesian Estimation of GARCH Models: Theory and Applications, Springer-Verlag, Berlin, 2008.

[3] R. Battiti, F. Masulli, BFGS optimisation for faster and automated supervised learning, in: Proceedings of the International Neural Networks Conference (INNC-90), vol. 2, Kluwer Academic Publ., 1990, pp. 757–760.

[4] L. Bauwens, C.M. Hafner, S. Laurent, Handbook of Volatility Models and Their Applications, John Wiley and Sons, Hoboken, NJ, 2012.

[5] M. Bildirici, O.O. Ersin, Improving forecasts of GARCH family models with the artificial neural networks, Expert Syst. Appl. 36 (4) (2009) 7355–7362.

[6] C. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, Oxford, UK, 1995.

[7] T. Bollerslev, Generalized autoregressive conditional heteroskedasticity, J. Economet. 31 (1986) 307–327.

[8] T. Bollerslev, E. Ghysels, Periodic autoregressive conditional heteroskedasticity, J. Business Econ. Stat. 14 (1996) 139–151.

[9] M.A. Carnero, D. Peña, E. Ruiz, Persistence and kurtosis in GARCH and stochastic volatility models, J. Finan. Economet. 2 (3) (2004) 319–342.

[10] G.C. Cawley, N.L.C. Talbot, R.J. Foxall, S.R. Dorling, D.P. Mandic, Heteroscedastic kernel ridge regression, Neurocomputing 57 (2004) 105–124.

[11] X. Cheng, P.L.H. Yu, W.K. Li, On a dynamic mixture GARCH model, J. Forecast. 28 (3) (2008) 247–265.

[12] M. Costa, A.L. Goldberger, C.-K. Peng, Multiscale entropy analysis of complex physiologic time series, Phys. Rev. Lett. 89 (6) (2002).

[13] P.J. Deschamps, Bayesian estimation of generalized hyperbolic skewed student GARCH models, Comput. Stat. Data Anal. 56 (2012) 3035–3054.

[14] R.G. Donaldson, M. Kamstra, An artificial neural network—GARCH model of international stock return volatility, J. Empirical Finan. 4 (1) (1997) 17–46.

[15] R.F. Engle, Autoregressive conditional heteroscedasticity with estimates of the variance of UK inflation, Econometrica 50 (1982) 987–1007.

[16] G. Fiorentini, G. Calzolari, L. Panattoni, Analytical derivatives and the computation of GARCH estimates, J. Appl. Economet. 11 (1996) 399–417.

[17] R. Fletcher, Practical Methods for Optimization, second ed., John Wiley and Sons, New York, 1987.

[18] B. Freisleben, K. Ripper, Volatility estimation with a neural network, in: Proceedings of the IEE/IAFE Computational Intelligence for Financial Engineering Conference (CIFEr-97), IEEE Press, 1997, pp. 177–181.

[19] V.V. Gavrishchaka, S.B. Ganguli, Volatility forecasting from multiscale and high-dimensional market data, Neurocomputing 55 (1–2) (2003) 285–305.

[20] I.A. Gheyas, L.S. Smith, A novel neural network ensemble architecture for time series forecasting, Neurocomputing 74 (18) (2011) 3855–3864.

[21] M. Haas, S. Mittnik, M.S. Paoella, Mixed normal conditional heteroskedasticity, J. Finan. Economet. 2 (2004) 211–250.

[22] L.U. Hjorth, Regularization in Mixture Density Networks, Technical Report NCRG/99/004, Aston University, Birmingham, UK, 1999.

[23] M.I. Jordan, R.A. Jacobs, Hierarchical mixtures of experts and the EM algorithm, Neural Comput. 6 (1994) 181–214.

[24] S. Kim, N. Shephard, S. Chib, Adaptive rejection metropolis sampling within Gibbs sampling, Appl. Stat. 44 (1998) 155–173.

[25] W. Li, J. Liu, J. Le, Using GARCH-GRNN model to forecast financial time series, in: Proceedings of the 20th International Conference on Computer and Information Sciences (ISCIS'05), Lecture Notes in Computer Science, vol. 3733, Springer-Verlag, Berlin, 2005, pp. 565–574.

[26] G.J. McLachlan, K.E. Basford, Mixture Models: Inference and Applications to Clustering, Marcel Dekker, New York, 1988.

[27] T. Miazhynskaia, G. Dorffner, E.J. Dockner, Risk management application of the recurrent mixture density network models, in: International Conference on Artificial Neural Networks ICANN-2003, Lecture Notes in Computer Science, vol. 2714, Springer, Berlin, 2003, pp. 589–596.

[28] D. Ormoneit, R. Neuneier, Experiments in predicting the German stock index DAX with density estimating neural networks, in: Proceedings of the IEEE/IAFE 1996 Conference on Computational Intelligence for Financial Engineering (CIFEr 96), New York, 1995, pp. 66–71.

[29] L. Pascual, J. Romo, E. Ruiz, Bootstrap prediction for returns and volatilities in GARCH models, Comput. Stat. Data Anal. 50 (2006) 2293–2312.

[30] F. Pérez-Cruz, J.A. Afonso-Rodriguez, J. Giner, Estimating GARCH models using support vector machines, J. Quant. Finan. 3 (3) (2003) 163–172.

[31] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning internal representations by error propagation, in: D.E. Rumelhart, J.L. McLelland (Eds.), Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1, The MIT Press, Cambridge, MA, 1986, pp. 318–362. (Chapter 8).

[32] C. Schittenkopf, G. Dorffner, E.J. Dockner, Volatility prediction with mixture density networks, in: L. Niklasson, M. Boden, T. Ziemke (Eds.), Proceedings of the Eighth International Conference on Artificial Neural Networks (ICANN-98), Lecture Notes in Computer Science, vol. 1240, Springer, Berlin, 1998, pp. 929–934.

[33] C. Schittenkopf, G. Dorffner, E.J. Dockner, Forecasting time-dependent conditional densities: a semi non-parametric neural network, J. Forecast. 19 (4) (2000) 355–374.

[34] H. Tang, K.C. Chiu, L. Xu, Finite mixture of ARMA-GARCH model for stock price prediction, in: Proceedings of the Third International Workshop on Computational Intelligence in Economics and Finance (CIEF'2003), North Carolina, USA, 2003, pp. 1112–1119.

[35] M. Uddin Ahmed, D.P. Mandic, Multivariate multiscale entropy: a tool for complexity analysis of multichannel data, Phys. Rev. E 84 (6) (2011).

[36] P. Werbos, Backpropagation through time: what it does and how to do it, Proc. IEEE 78 (1990) 1550–1560.

[37] R.J. Williams, D. Zipser, A learning algorithm for continually running fully recurrent networks, Neural Comput. 1 (1989) 270–280.

[38] R.J. Williams, D. Zipser, Gradient-based learning algorithms for recurrent networks and their computational complexity, in: Y. Chauvin, D.E. Rumelhart (Eds.), Back-Propagation: Theory, Architectures and Applications, Lawrence Erlbaum Publ., Hillsdale, NJ, 1995, pp. 433–486.
[39] C.S. Wong, W.K. Li, On a logistic mixture autoregressive model, Biometrika 88 (3) (2001) 833–846.

Nikolay Nikolaev received the PhD degree in Computer Science and Engineering from the Technical University, Sofia, Bulgaria, in 1992. From 1992 to 1993, he conducted postdoctoral research in machine learning at the University of Wales, Cardiff, UK. From fall 1993, he was a Lecturer in Computer Science at the American University, Bulgaria. In fall 2000, he joined the Department of Mathematical and Computing Sciences, Goldsmiths College, University of London, London, UK, as a Lecturer in Computing. In 2000 and 2001, he was a Research Fellow with the evolutionary computation group at The University of Tokyo. His theoretical research interests include the following: neural networks sequential Bayesian inference, stochastic volatility, dynamic nonlinear GARCH modeling. His current application interests include: financial value-at-risk estimation, portfolio allocation, volatility arbitrage and statistical pairs trading.

Peter Tino (MSc, Slovak University of Technology; PhD, Slovak Academy of Sciences) was a Fulbright Fellow with the NEC Research Institute, Princeton, NJ, USA, and a Post-Doctoral Fellow with the Austrian Research Institute for AI, Vienna, Austria, and with Aston University, Birmingham, UK. Since 2003, he has been with the School of Computer Science, University of Birmingham, Edgbaston, Birmingham, UK, where he is currently a Reader in complex and adaptive systems. His current research interests include dynamical systems, machine learning, probabilistic modeling of structured data, evolutionary computation, and fractal analysis. Peter was a recipient of the Fulbright Fellowship in 1994, the UK and Hong-Kong Fellowship for Excellence in 2008, three Outstanding Paper of the Year Awards from the IEEE Transactions on Neural Networks in 1998 and 2011 and the IEEE Transactions on Evolutionary Computation in 2010, and the Best Paper Award at ICANN 2002. He serves on the editorial boards of several journals.

Evgueni Smirnov is an Assistant Professor of Computer Science at the Department of Knowledge Engineering, Maastricht University, The Netherlands. He graduated in Computer Science from the Technical University of Sofia, Bulgaria, in 1988, and he earned his PhD degree in Artificial Intelligence from Maastricht University in 2001. His theoretical research interests are in the machine-learning field: meta-learning, ensemble learning, reliable prediction, kernel methods, version spaces, and complexity. His current application interests include: applications of data mining in medicine, transportation, and machinery-automation.