Recurrent Neural Networks with more flexible memory: better predictions than rough volatility

Damien Challet, Vincent Ragel

RECURRENT NEURAL NETWORKS WITH MORE FLEXIBLE MEMORY: BETTER PREDICTIONS THAN ROUGH VOLATILITY

Damien Challet[1] **and Vincent Ragel**[1,2]
[1] Université Paris Saclay, CentraleSupélec, Laboratoire MICS,
91190 Gif-sur-Yvette, France
[2] BNP Paribas, 20 boulevard des Italiens, 75008 Paris, France
damien.challet,vincent.ragel@centralesupelec.fr

ABSTRACT

We extend recurrent neural networks to include several flexible timescales for each dimension of their output, which mechanically improves their abilities to account for processes with long memory or with highly disparate time scales. We compare the ability of vanilla and extended long short term memory networks (LSTMs) to predict asset price volatility, known to have a long memory. Generally, the number of epochs needed to train extended LSTMs is divided by two, while the variation of validation and test losses among models with the same hyperparameters is much smaller. We also show that the model with the smallest validation loss systemically outperforms rough volatility predictions by about 20% when trained and tested on a dataset with multiple time series.

***K*eywords** Time series · Long memory · Recurrent Neural Networks · Rough Volatility · Volatility modelling

1 Introduction

Some time series in Nature have a very long memory (Robinson, 2003): fluid turbulence (Resagk et al., 2006), asset price volatility (Cont, 2001) and tick-by-tick events in financial markets (Challet and Stinchcombe, 2001; Lillo and Farmer, 2004). From a modelling point of view, this means that the current value of an observable of interest depends on the past by a convolution of itself with a long-tailed kernel.

Deep learning tackles past dependence in time series with recurrent neural networks (RNNs). These networks are in essence moving averages of nonlinear functions of the inputs and learn the parameters of these averages and functions. Provided that they are sufficiently large, these networks can approximate long-tailed kernels in a satisfactory way, and are of course able to account for more complex problems than a simple linear convolution. Yet, their flexibility may prevent them to learn quickly and efficiently the long memory of time series. Several solutions exist: either one pre-filters the data by computing statistics at with various time scales and use them as inputs to RNNs in the same spirit as multi-time scale volatility modelling (Zumbach and Lynch, 2001; Corsi, 2009), see e.g. Kim and Won (2018), or one extends the neural networks so as to improve their abilities. For example, Zhao et al. (2020) adds delay operators, taking inspiration from the ARIMA processes, to the states of recurrent neural networks, while Ohno and Kumagai (2021) modifies the update equation of the network output so that its dynamics mimics that of a variable with a long memory. In both cases, the time dependence structure is enforced by hand in the dynamics of such networks.

Here, we propose a flexible and parsimonious way to extend the long-memory abilities of recurrent neural networks by using an old trick for approximating long-memory kernels with exponential functions, which helps recurrent neural networks learn faster and better time series with long memory.

Our main contributions are: (i) we introduce RNNs with several multiple flexible time scales for each dimension of the output; (ii) we show that learning to predict time series with long-memory (asset price volatility) is faster and more reliably good with more flexible time scales (iii) rough volatility predictions can be beaten by training a fair number of recurrent neural networks and only using the one with the best validation loss.

2 Methods

Let time series y_t be of interest. Its moving average can be written

$$\tilde{y}_t = \int_{-\infty}^{t} K(t-t')y'_t dt', \tag{1}$$

where K is a kernel. In a discrete time context,

$$\tilde{y}_t = \sum_{-\infty}^{t} K(t-t')y'_t. \tag{2}$$

When the process is Markovian, its kernel $K(x) \simeq e^{-x/\tau_0}$ for large x, where τ_0 is the slowest timescale at which the process forgets its past. In this case, one can write y_t in a recursive way

$$\tilde{y}_t = \tilde{y}_{t-1}(1-\lambda) + \lambda y_t, \tag{3}$$

where $\lambda \simeq 1/\tau_0$; $\tilde{y}_t$ is then an exponentially moving average (EMA) of y_t.

Long memory processes however, have a kernel that decreases at least as slowly as a power-law. In turn, power-laws can be approximated by a sum of exponential functions: naively, if $K(x) = x^{-\alpha}$, one writes

$$K(x) \propto \sum_{i=1}^{\infty} w_i \exp(-x/\tau_i) \tag{4}$$

with $w_i \propto (1/c^\alpha)^i$ and $\tau_i = c^i$ for a well-chosen constant c: one covers the x space in a geometric way and the weights w_i account for the power-law decreasing nature of $K(x)$. This rough approach works well and is widespread. Bochud and Challet (2007) propose a more refined method to determine how many exponential functions one needs to approximate optimally K and how to compute w_i for a given α and for a given range of x over which the kernel has to be approximated by a sum of exponential functions (e.g. $x \in [1, 1000]$). For example, one needs about 4 exponential functions to approximate 3 decades).

Writing down the update equations of well-known recurrent neural network architectures makes it clear that they use exponentially moving averages with a single time scale for each output dimension. For example, Gate Recurrent Units (GRU) Cho et al. (2014) transform the input vector x_t into a vector of timescales λ_t defined as

$$\lambda_t = \sigma(W_\lambda x_t + U_\lambda c_{t-1} + b_\lambda) \tag{5}$$

which is then used in the update of the output c_t

$$c_t = c_{t-1} \odot (1-\lambda) + \lambda \odot \tilde{c}_t, \tag{6}$$

where the update $\tilde{c}_t$ is also computed from the input with learned weights, i.e.

$$\tilde{c}_t = \sigma_c(W_c x_t + U_c(c_{t-1} \odot r_t) + b_c) \tag{7}$$

$$r_t = \sigma_r(W_r x_t + U_r c_{t-1} + b_r) \tag{8}$$

for a non-linear functions σ_c and σ_r, $\odot$ is the element-wise (Hadamar) product and r_t the reset gate which modifies the value of c_{t-1} when computing $\tilde{c}_t$. By design, GRUs can only compute exponentially moving averages of $\tilde{c}_t$, although they possess the interesting ability to learn both λ_t and the update $\tilde{c}_t$ as a function of their inputs. It is straightforward to extend GRUs to an arbitrary number of timescales n by using n $\tilde{c}_t^{(k)}$, $k = 1, \cdots, n$ and

$$\lambda_t^{(k)} = \sigma(W_\lambda^{(k)} x_t + U_\lambda^{(k)} c_{t-1}^{(k)} + b_\lambda^{(k)}) \tag{9}$$

$$c_t^{(k)} = c_{t-1}^{(k)} \odot (1-\lambda_t^{(k)}) + \lambda_t^{(k)} \tilde{c}_t, \tag{10}$$

where each $c_t^{(k)}$ is an exponentially moving average at time scale $\sim 1/\lambda_t^{(k)}$. Finally, the output will be

$$c_t = \sum_{k=1}^{n} w_k c_t^{(k)}, \tag{11}$$

The simple $\alpha-$RNN (Dixon and London, 2021), which are simplified GRUs, share the same assumption of a single time scale per output dimension and thus can be generalized in the same way. Let us show now how extend LSTMs with a forget gate (Gers et al., 2000). Starting from their output h_t, one has

$$h_t = o_t \odot \sigma_h(c_t) \tag{12}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \tag{13}$$

where o_t, i_t, and $\tilde{c}_t$ are determined from the input x_t and the previous output h_{t-1} with learned weights and σ_h is a nonlinear function. Writing $f_t = 1 - \lambda_t$ makes it obvious that the cell vector c_t evolves in the same way as y_t in Eq. (6) if $i_t \simeq \lambda_t$.

Extending LSTMs to include n time scales by cell state dimension is therefore straightforward: one needs to compute n EMAs and their associated λs as follows

$$f_t^{(k)} = \sigma(W_f^{(k)} x_t + U_f^{(k)} h_{t-1} + b_f^{(k)}) \tag{14}$$
$$i_t^{(k)} = \sigma(W_i^{(k)} x_t + U_i^{(k)} h_{t-1} + b_i^{(k)}) \tag{15}$$
$$c_t^{(k)} = f_t^{(k)} \odot c_{t-1}^{(k)} + i_t^{(k)} \odot \tilde{c}_t, \tag{16}$$

where $\tilde{c}_t$ follows Eq. (11). Note that one could set $i_t^{(k)} = 1 - f_t^{(k)} = \lambda_t^{(k)}$ and not learn the weights associated to i_t. Learning as well $i^{(k)}$ is equivalent to modulate the importance of the update, which is known as to as cognitive bias (Palminteri et al., 2017): this is made clear by writing $i_t^{(k)} = v_t^{(k)}(1 - f_t^{(k)}) = v_t^{(k)} \lambda_t^{(k)}$, where $v_t^{(k)}$ is the modulation of learning speed.

We will focus on the case $n = 2$: (11) amounts to

$$c_t = c_t^{(1)} \odot \alpha + (1 - \alpha) \odot c_t^{(2)}, \tag{17}$$

where the vector α is learnable and its components are bounded to the $[0, 1]$ interval. We call LSTMs with several time scales ($n > 1$) per dimension VLSTMs, which stands for very long short term memory.

Note that LSTMs with a sufficiently large cell dimension N_h can in principle learn to superpose timescales in the same way as Eq. (11) by learning one time scale per dimension and using final dense layer to learn how to combine them. However, imposing constraints (or equivalently, injecting some known structure) is known to lead to faster learning and better results (e.g. Physics-guided deep learning, see Thuerey et al. (2021) for a review).

Naively, when learning to predict a process that is not too noisy, we expect the difference between VLSTM and LSTM to be the highest when $N_h = 1$, i.e. precisely when LSTMs do not have the possibility to compute long-term averages and to decrease when N_h increases.

2.1 Volatility prediction

Given an asset price P_t, its and its log return $r_t = \log P_t - \log P_{t-1}$, the asset price volatility σ is defined as $\sigma^2 = E(r^2)$. The dynamics of financial markets is ever-changing, which results in a temporal dependence of σ with clear patterns of long-term dependence (Cont, 2001) (see Fig. 1).

Risk management, portfolio optimization. and option pricing benefit from the ability to predict the evolution of σ_t. Fortunately, σ_t is relatively easy to predict, owing to its long memory (Cont, 2001): for example, its auto-correlation decreases very slowly, presumably as a power-law over more than a year. Econometric models include GARCH, whose simplest version involves only one timescale, while many variations use several time scales (Zumbach and Lynch, 2001; Corsi, 2009; Zumbach, 2015). Rough volatility (Gatheral et al., 2018), on the other hand, considers $\log \sigma_t$ as fractional Brownian motion and thus includes all the time scales. As can be expected, rough volatility models outperform GARCH-based models for volatility prediction. Using LSTMs for volatility prediction is found e.g. in Kim and Won (2018); Filipović and Khalilzadeh (2021); Rosenbaum and Zhang (2022) that use various types of predictors (including GARCH models) and architectures. Notably, Rosenbaum and Zhang (2022) show that the average prediction of 10 stacked LSTMs with past volatility and price return as predictors match the performance of rough volatility.

2.2 Architecture and hyperparameters

Our first aim is to characterise the effects of multiple time scales per cell dimension. Therefore, we compare simple non-stacked LSTMs with or without the proposed modification. Stacked LSTMs can learn additional time scales at

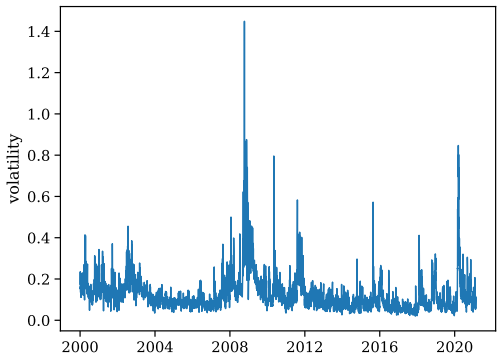

Figure 1: Volatility of the SPX index (annualized) as a function of time, showing traces of long-term dependence. Data source: Oxford-Man Institute (Heber et al., 2022).

the cost of doubling the number of parameters, which we precisely wish to avoid here. We pass the outputs h_t of the LSTMs and VLSTMs through a dense layer of size N_h with sigmoid activation functions, so as to combine the outputs in a non-trivial way, and a final dense layer with linear activation. Both final layers have a bias term, which allows the model to learn a baseline volatility level.

We report a systematic study of the relative performance of LSTMs vs VLSTMs. We vary the sequence length T_{seq} from 10 to 100 by steps of 15, and the dimension of the hidden state $N_h \in \{1, \cdots, 5\}$. Finally, we train models with and without biases (except for the final two dense layers which always have biases). There are thus 70 variations of hyperparameters per architecture choice.

For each hyperparameter and architecture couple, we train 20 networks which yields 2800 models altogether. We use a standard 60/20/20 train/validation/test splits and apply early stopping criterion of the minimum validation loss over the 5 last epochs, with a maximum of 1000 epochs. Batch size is set to 128. We train the networks to predict $\log \sigma_{t+1}$ with an MSE loss function.

Data comes from volatility computed by Oxford-Man Institute from intraday data with the two-scale realized kernel estimation method (Barndorff-Nielsen et al., 2008), which contain volatility time series for 31 indices and 2117 to 5385 data points per index (Heber et al., 2022). Since the volatility individual time series start and end at heterogeneous dates, we used the dates to define the train/validation/test splits: the train set ranges from 2000-01-04 to 2012-09-06, validation set from 2012-09-07 to 2016-11-23 and test set from 2016-11-24 to 2021-02-17. This is necessary as the time series are cross-correlated, hence, splitting them according to their respective length would cause information leakage from the future and thus overfitting.

3 Results

3.1 Average loss

Let us plot the average test loss of LSTMs and VLSTMs as a function of N_h at fixed T_{seq}, the dimension of the memory cell, and of T_{seq} at fixed N_h. This approach is taken by Rosenbaum and Zhang (2022) who trained 10 LSTMs instead of 20 here. Figure 2 shows that VLSTMs enjoy a sizeable advantage on average. We note that when $N_h = 1$, our initial intuition was correct: VLSTMs have a smaller average test loss for all variations of hyperparameters (T_{seq} and bias)

Large loss fluctuations among models are associated with large average test losses for both VLSTMs and LSTMs; however test losses of VLSTMs are more likely to be small (and have accordingly small fluctuations). This is explained by a large difference in training convergence time, as shown below. We also note that, at least for volatility prediction, keeping bias terms in the computation of i, f, $\tilde{c}$, and c (referred to as internal biases henceforth) is manifestly problematic; it turns out to be the default option both for PyTorch and Keras and is probably implicitly used in other papers. On the whole, we note that a simple average of the outputs of an ensemble of models leads to quite large fluctuations, hence that the question of the convergence of the models must be investigated and a way to select the good models would much improve the usefulness of LSTMs in that context.

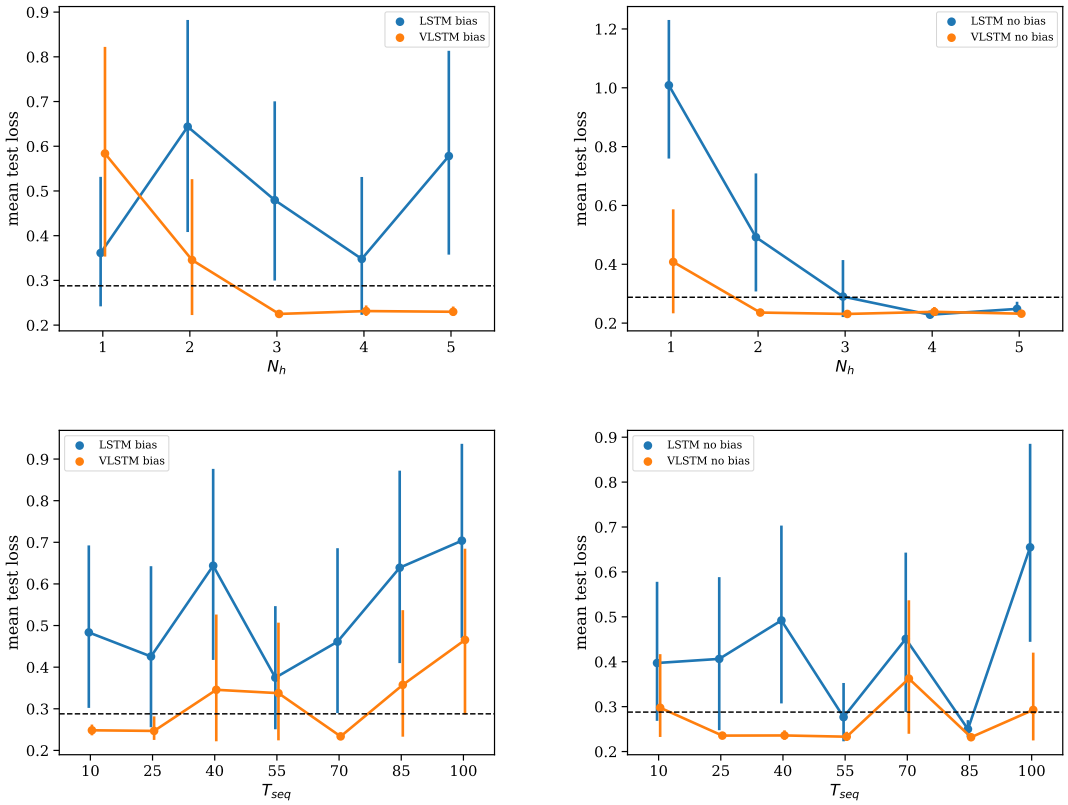

Figure 2: Volatility prediction. Upper plots: mean test loss vs the memory cell dimension N_h ($T_{\text{seq}} = 40$); lower plots: mean test loss vs the sequence length T_{seq} ($N_h = 2$). Left plots: (V)LSTMs with bias weights; right plots: (V)LSTMs with no bias weights. The dashed line is the average MSE of predictions made with rough volatility models.

Train convergence, it turns out, is a hit and miss process: some models are stuck in a high loss regime, while some models do learn a more realistic dynamical process and reach much lower losses. This yields a bi-modal density of losses (see Fig. 3). It is noteworthy that the fraction of VLSTMs that learn better is much larger. This is linked to the fact that VLSTMs learn much faster (see below) and that VLSTMs without internal biases are less likely to be stuck in a high loss regime.

Since the volatility process is well approximated e.g. by a rough volatility model (**?**), the test loss is commensurate with the validation loss as expected, itself commensurate with the train loss. We plot in Fig. 3 the test loss versus the validation loss, which shows that test losses are accordingly also bimodal, with a majority of models not stuck in the high loss regime, some having a test loss smaller than rough volatility models.

3.2 Keeping the better models

This result suggests a way to select the good models, since the validation loss distribution is bimodal and since the test losses are roughly proportional to validation losses. To select models whose validation loss belongs in the lower peak, we compute 9 quantiles $q(p)$ with regular sequence of probabilities $p = 0.1, \cdots, 0.9$, and keep the models whose validation loss is smaller than the quantile corresponding to the maximum change between quantiles, a simple yet effective way to find well separated peaks. We call these models the better ones in the following. This procedure allows a fairer comparison between LSTMs and VLSTMs.

VLSTMs are still better than LSTMs, even for larger N_h. Figure 4 plots the average test loss of the models with below-average validation loss versus N_h and the sequence length. The test losses are now much closer, but VLSTMs still retain a sizable advantage: their test losses are both lower on average and their fluctuations are much smaller.

Both the variability of results and the strange results for $N_h = 1$ when biases are allowed in the computation of the internal states of (V)LSTMs can be traced back to training convergence problems. A simple way to ascertain the main

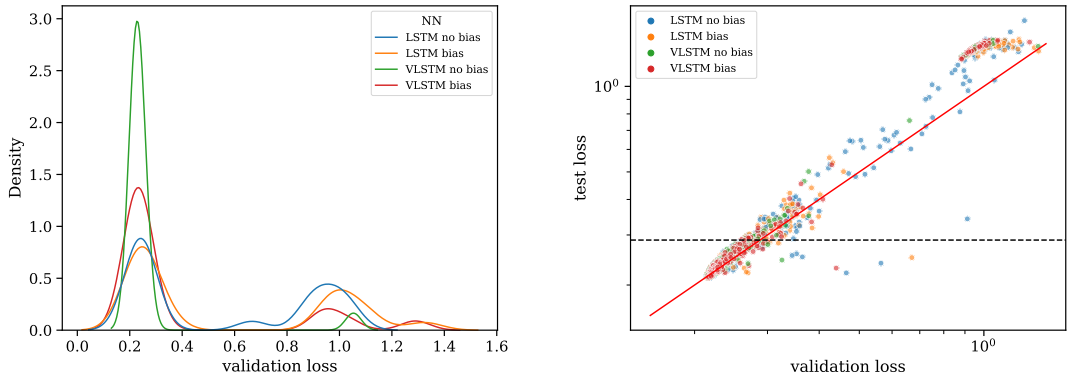

Figure 3: Left plot: density of validation losses by architecture. Right plot: test loss vs validation loss. Multiple volatility time series prediction. The dashed line is the average MSE of predictions made with rough volatility models.

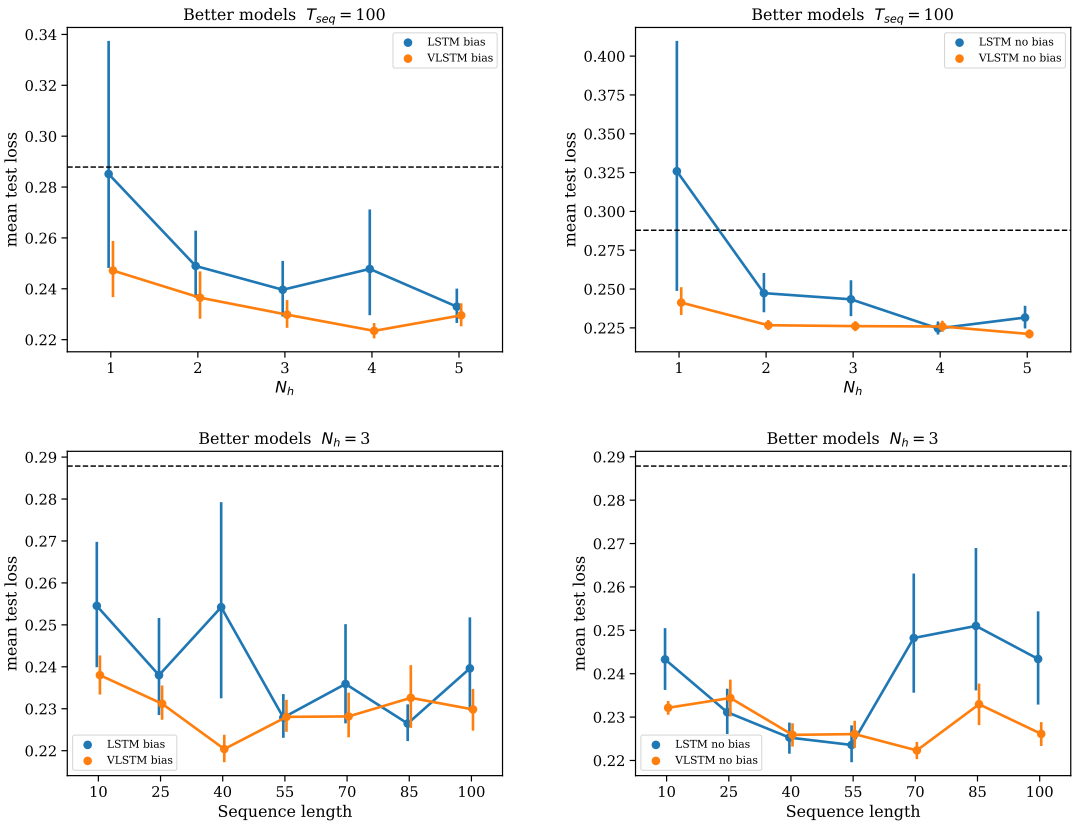

Figure 4: Multiple volatility time series prediction test losses of the models with below-average validation losses. Upper plots: mean test loss vs the memory cell dimension N_h ($T_{\text{seq}} = 100$); lower plots: mean test loss vs the sequence length T_{seq} ($N_h = 3$). Left plots: (V)LSTMs with bias weights; right plots: (V)LSTMs with no bias weights. The dashed line is the average MSE of predictions made with rough volatility models.

Architecture	bias	test loss average	test loss std dev.
rough vol.		0.288	
LSTM	yes	0.241	0.032
LSTM	no	0.245	0.057
VLSTM	yes	0.232	0.017
VLSTM	no	0.230	0.015

Table 1: Standard deviation of the test losses of the better models computed over all the values of N_h and T_{seq}. Multiple time series volatility prediction

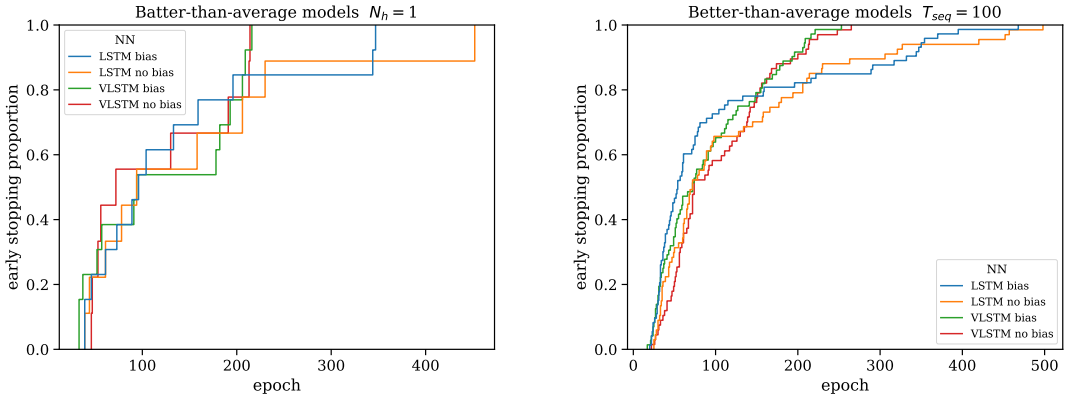

Figure 5: Fraction of models having converged before a given number of epochs. Left plot: $N_h = 1$, right plot: all N_h; multiple time series volatility prediction, $T_{\text{seq}} = 100$.

difference between VLSTMs and LSTMs is to measure the time it takes for their training to converge, i.e., to reach the early stopping criterion. Figure 5 reports the fraction of models that have converged as a function of the number of epochs (limited to 1000). LSTMs need more epochs to be trained. We also found that the case $N_h = 1$ and small T_{seq} is hard to learn for this kind of architecture, the training of many models requiring more than 1000 epochs to reach the early stopping criterion.

3.3 Best model

Finally, let us investigate the test loss of the model with the best validation loss among the 20 models trained for each of the 140 hyperparameters/architecture choices. It turns out that under these conditions VLSTMs and LSTMs have essentially the same performance. What differentiate them however is the speed at which they learn. Let us plot the test loss versus the time of convergence for LSTMs and VLSTMs with and without biases (left plot of Fig. 6): there is slight negative dependence between test loss and convergence times, the longer one learns, the better. Notably convergence times of LSTMs are spread all over the whole $[1, 1000]$ interval, while VLSTMs converge before 400 epochs. The right plot of Fig. 6 displays the ECDF of the convergence times, which shows a sizable difference between LSTMs and VLSTMs: whereas 20% of LSTMs models do not manage to converge before 1000 epochs, all VLSTMs do before 400, except one, when biases are allowed.

Thus, training a given number of models is significantly shorter with VLSTMs because they do not need to learn how to approximate the kernel $K(x)$. One also sees that models with internal biases converge more slowly than those without them. We also wish to point out that because the fluctuation of validation losses among the trained models is much smaller for VLSTMs than for LSTMs, hence, that in practice, one needs to train fewer VLSTMs than LSTMs before finding a good one.

4 Conclusion

Adding an explicit but flexible kernel structure to LSTMs brings significant improvements in every metric: number of epochs needed to reach convergence, overall prediction accuracy, and accuracy variation between models at fixed

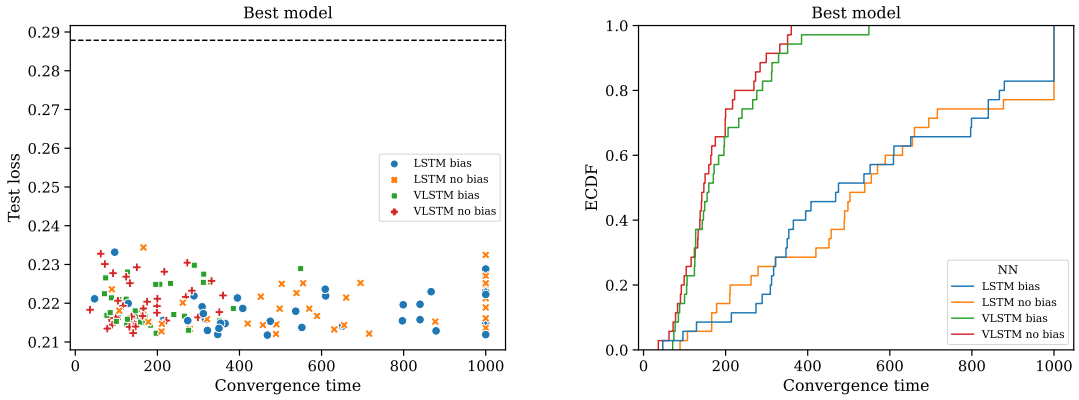

Figure 6: Left plot: test loss of the models with the best validation loss for all architecture and hyperparameter choices. Right plot: empirical cumulative distribution function of the convergence time for the four architecture choices. All values of N_h and T_{seq}; multiple time series volatility prediction. The dashed line is the average MSE of predictions made with rough volatility models.

parameters. There is a cost as the number of trainable parameters of VLSTMs is larger than LSTMs are fixed hyperparameters, but doubling the number of time scales does not require to double the number of trainable parameters, thanks to the explicit kernel approximation structure. Although this paper focuses on LSTMs, the same idea can be applied to GRUs and $\alpha-$RNNs in a straightforward way.

Our results mirror those of Rosenbaum and Zhang (2022): we also succeeded in training a single model at a time on many volatility time series of various underlying asset types. This reflects the universality of volatility dynamics, a fact also hinted at by rough volatility and multi-scale GARCH-like models (Zumbach, 2015).

While it is hard to beat rough volatility for volatility prediction, we found that even simple LSTMs can beat it, provided that one trains several models and selects the best one according to its validation loss. Using LSTMs for that purpose requires to train more models over more epochs than using VLSTMs. Volatility prediction can be further improved by adding some more features, such as prior knowledge of predictable special events, and possibly by using more complex neural architectures.

5 Code and data availability

Full code, including the Keras VLSTM class, and data, are available at `https://github.com/damienchallet/VLSTM`.

Acknowledgments

This work used HPC resources from the "Mésocentre" computing center of CentraleSupélec and École Normale Supérieure Paris-Saclay supported by CNRS and Région Île-de-France.

References

Ole E Barndorff-Nielsen, Peter Reinhard Hansen, Asger Lunde, and Neil Shephard. Designing realized kernels to measure the ex post variation of equity prices in the presence of noise. *Econometrica*, 76(6):1481–1536, 2008.

Thierry Bochud and Damien Challet. Optimal approximations of power laws with exponentials: application to volatility models with long memory. *Quantitative Finance*, 7(6):585–589, 2007. doi: 10.1080/14697680701278291.

Damien Challet and Robin Stinchcombe. Analyzing and modeling 1+ 1d markets. *Physica A: Statistical Mechanics and its Applications*, 300(1-2):285–299, 2001.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Rama Cont. Empirical properties of asset returns: stylized facts and statistical issues. *Quantitative finance*, 1(2):223, 2001.

Fulvio Corsi. A simple approximate long-memory model of realized volatility. *Journal of Financial Econometrics*, 7 (2):174–196, 2009.

Matthew Dixon and Justin London. Financial forecasting with α-rnns: A time series modeling approach. *Frontiers in Applied Mathematics and Statistics*, 6, 2021. doi: 10.3389/fams.2020.551138.

Damir Filipović and Amir Khalilzadeh. Machine learning for predicting stock return volatility. *Swiss Finance Institute Research Paper*, (21-95), 2021.

Jim Gatheral, Thibault Jaisson, and Mathieu Rosenbaum. Volatility is rough. *Quantitative finance*, 18(6):933–949, 2018.

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with LSTM. *Neural computation*, 12(10):2451–2471, 2000.

Gerd Heber, Asger Lunde, Neil Shephard, and Kevin Sheppard. Oxford-Man Institute's realized library, 2022. URL `https://realized.oxford-man.ox.ac.uk/`.

Ha Young Kim and Chang Hyun Won. Forecasting the volatility of stock price index: A hybrid model integrating lstm with multiple GARCH-type models. *Expert Systems with Applications*, 103:25–37, 2018. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2018.03.002.

Fabrizio Lillo and J Doyne Farmer. The long memory of the efficient market. *Studies in Nonlinear Dynamics & Econometrics*, 8(3), 2004.

Kentaro Ohno and Atsutoshi Kumagai. Recurrent neural networks for learning long-term temporal dependencies with reanalysis of time scale representation. In *2021 IEEE International Conference on Big Knowledge (ICBK)*, pages 182–189. IEEE, 2021.

Stefano Palminteri, Germain Lefebvre, Emma J Kilford, and Sarah-Jayne Blakemore. Confirmation bias in human reinforcement learning: Evidence from counterfactual feedback processing. *PLoS computational biology*, 13(8): e1005684, 2017.

Christian Resagk, Ronald du Puits, André Thess, Felix V Dolzhansky, Siegfried Grossmann, Francisco Fontenele Araujo, and Detlef Lohse. Oscillations of the large scale wind in turbulent thermal convection. *Physics of fluids*, 18(9): 095105, 2006.

Peter M Robinson. *Time series with long memory*. Advanced Texts in Econometrics, 2003.

Mathieu Rosenbaum and Jianfei Zhang. On the universality of the volatility formation process: when machine learning and rough volatility agree. *arXiv preprint arXiv:2206.14114*, 2022.

Nils Thuerey, Philipp Holl, Maximilian Mueller, Patrick Schnell, Felix Trost, and Kiwon Um. *Physics-based Deep Learning*. WWW, 2021. URL `https://physicsbaseddeeplearning.org`.

Jingyu Zhao, Feiqing Huang, Jia Lv, Yanjie Duan, Zhen Qin, Guodong Li, and Guangjian Tian. Do RNN and LSTM have long memory? In *International Conference on Machine Learning*, pages 11365–11375. PMLR, 2020.

Gilles Zumbach. Cross-sectional universalities in financial time series. *Quantitative Finance*, 15(12):1901–1912, 2015.

Gilles Zumbach and Paul Lynch. Heterogeneous volatility cascade in financial markets. *Physica A: Statistical Mechanics and its Applications*, 298(3-4):521–529, 2001.