From Deep Learning to Deep Econometrics

Robert Stok, Paul Bilokon, and Joseph Simonian

Robert Stok
is a researcher at Imperial College in London, UK.
rs.stok@gmail.com

Paul Bilokon
is the chief executive officer of Thalesians, Ltd., and a visiting lecturer at Imperial College in London, UK.
paul.bilokon01@imperial.ac.uk

Joseph Simonian
is the founder and CIO of Autonomous Investment Technologies LLC in Newton, MA.
jsslog15@gmail.com

KEY FINDINGS

- The authors implement a hybrid particle filter and recurrent neural network model for volatility filtering.
- They introduce randomness to the input of the neural network to allow it to learn nonlinear noise distributions in its transition function.
- They modify the loss function to penalize outlier particles less so that the model can explore.

ABSTRACT

Calculating true volatility is an essential task for option pricing and risk management. It is made difficult, however, by market microstructure noise. Particle filtering has been proposed to solve this problem because it has favorable statistical properties, but it relies on assumptions about underlying market dynamics. Machine learning methods have also been proposed but lack interpretability and often lag in performance. In this article, the authors implement the stochastic volatility (SV)-PF-RNN: a hybrid neural network and particle filter architecture. Their SV-PF-RNN is designed specifically with stochastic volatility estimation in mind. They demonstrate that it can significantly improve the performance of a basic particle filter.

Econometrics and machine learning are each concerned with the processing of statistical information. Their historical and theoretical roots are quite different however. The methodological foundations of contemporary econometrics were laid amid a debate that was epitomized by Tjalling Koopmans' critical review (Koopmans 1947) of Arthur Burns and Wesley Mitchell's book, *Measuring Business Cycles* (Burns and Mitchell 1946). In the review, "Measurement without Theory," Koopmans, who was a member of the Cowles Commission, argued that economic data cannot be properly interpreted without the benefit of theoretically sound economic assumptions. The primary target of the review was the empiricist econometric methodology employed by the National Bureau of Economic Research (NBER), which Burns and Mitchell were associated with. Koopmans felt the NBER was overly preoccupied with devising tools for measuring economic data at the expense of the theory development necessary for drawing robust economic conclusions. Ultimately, Koopmans and the supporters of his methodology emerged victorious and set the tone for econometrics for the next half century.[1]

The rift between traditional econometrics and more purely empirical approaches accelerated with the advent of data science. Indeed, the rift became so large that

[1] See Simonian and Fabozzi (2019) for more on this debate.

Breiman (2001) declared that the "data modeling culture" represented by econometrics was starkly different from the "algorithmic modeling culture" represented by data science, while arguing that the latter was more practically useful than the former. The data modeling culture, which has been the dominant culture in traditional statistical practice, is primarily concerned with providing information about the relationships that exist between input and response variables. It assumes that the data are generated by a specific stochastic process. In contrast, the algorithmic modeling culture assumes that the relationships between input and response variables are ultimately too complex to uncover with any genuine insight. Consequently, the primary concern of algorithmic methods is the development of tools that can be used to successfully predict responses from inputs, without any a priori assumptions on the data distributions. By contrast, while econometric models are typically good at explaining the past, they are relatively poor at forecasting.[2]

Although the methodological differences between econometrics and data science are real, it is nevertheless possible to build hybrid models that draw on the strengths of each paradigm. An example of the latter type of hybridization is the framework known as *modular machine learning*, introduced by Simonian (2020). In the latter framework, econometrics (e.g., a regime-switching model) is used to divide economic data into different folds (subsamples). Machine learning is then used to train and validate a given model using these regime-specific folds. The advantage of this approach is that it uses econometrics to divide history into economically meaningful folds, a consideration that is ignored by standard approaches to cross-validation. The approach is modular because each component (module) of the framework is completely separate but inherits its input from another.

This article presents a hybrid approach to modeling an important market phenomenon: volatility. Rather than presenting a modular framework, the modeling approach described in this article can be best described as integrated. Volatility forecasting plays a fundamental role in finance and risk management, particularly in options pricing (Black and Scholes 1973; Ball and Roma 1994) and risk assessment (Shephard and Andersen 2009). Although volatility itself is not directly observable, it can be estimated by analyzing related time series, such as price shocks of correlated financial assets (Tsay 2010). Traditionally, volatility estimation has relied on autoregressive conditional heteroskedasticity (ARCH)/generalized ARCH (GARCH) models, which have demonstrated effectiveness (Bollerslev 1986; Hansen and Lunde 2005). However these models have limitations in that they do not account for exogenous variables and fail to capture certain characteristics of volatility, such as long memory, jumps, and leverage effects (Zivot 2009; Chan and Grant 2016).

Recently, researchers have explored the use of machine learning methods for volatility forecasting (Ge et al. 2022). Hybrid approaches combining machine learning and traditional statistical methods have shown promise (Christensen, Siggaard, and Veliyev 2023), with recurrent neural network (RNN) models delivering notable results because of their ability to capture time-series dependencies (Nguyen et al. 2022). A common criticism of machine learning methods, however, is their lack of interpretability (Christensen, Siggaard, and Veliyev 2023). Moreover, as functional rather than probabilistic models, they do not provide a comprehensive understanding of the distribution of estimated volatility, which is crucial for robust risk modeling. Another popular approach to modeling volatility is through stochastic volatility models, which employ Monte Carlo methods to approximate the time-varying probability distribution (Shephard 2005), accounting for changes in underlying stock prices. Particle filters (PFs)

[2] See Simonian and Wu (2019) for an in-depth discussion of this point with respect to regime-switching models.

are commonly used to approximate this distribution (Pitt and Shephard 1999; Malik and Pitt 2011).

In this article, our objective is to adapt the PF-RNN model proposed by Karkus, Hsu, and Lee (2018) to the task of volatility filtering. We call this the SV-PF-RNN. This architecture is a fully differentiable PF, providing the advantages of inference while generating a set of particles that approximates the likelihood of volatility.

Subsequently, we aim to demonstrate that our model can achieve or surpass the performance of a PF in estimating volatility from generated sequences. By generating volatility sequences and corresponding sets of returns using a stochastic volatility model, we train our hybrid model and evaluate its performance against the PF.

Finally, we undertake a thorough analysis of the limitations inherent in the SV-PF-RNN model. We identify specific challenges and propose potential remedies to address these shortcomings, while also highlighting future avenues for development of the SV-PF-RNN model.

BACKGROUND

This section is split into three main subsections. In the next two subsections, we will go over previous work that has been done on volatility prediction. In particular, we separately look at work that has focused on machine learning methods and work that has focused on particle filtering methods. In the third subsection, we look at work that has been done on combining PFs and neural networks, although this work has not yet been applied to the field of econometrics.

Volatility Prediction with Neural Networks

Standard neural networks. There is a large body of work that suggests combining traditional statistical methods with machine learning methods can yield better results than either of the two in isolation. One way to do this is to assume that volatility has both a linear and nonlinear component and then separately model these with statistical and neural network approaches. In his 2003 article, Zhang (2003) uses an autoregressive integrated moving average (ARIMA) model to forecast volatility and then trains a neural network to predict on the residuals. Although the hybrid model manages to outperform the ARIMA model, no information criterion analysis is done. Christensen, Siggaard, and Veliyev (2023) compare an extended heterogeneous autoregressive (HAR) model to three neural networks of increasing size, on predicting one-day-out volatility for 29 different stocks on the Dow Jones index. They find that the neural networks outperform the HAR model with up to an 11.8% reduction in the mean squared error (MSE) on certain financial instruments. They also address the common criticism that machine learning models are not interpretable by running an accumulated local effects analysis, allowing them to see which factors contribute most heavily to the network's prediction.

RNNs. RNNs are a popular choice for many time-series forecasting tasks because of their ability to infer patterns in long sequences of data (Ge et al. 2022). Purely RNN-based methods, however, have varying results and are often beaten by statistical methods in common time-series prediction tasks (Makridakis, Spiliotis, and Assimakopoulos 2018). Yang (Liu 2019) compared GARCH, v-support vector regression (v-SVR), and long-short term memory network (LSTM) models on three-day-out volatility prediction. He found that, although LSTMs outperformed the GARCH model, they had comparable performance to v-SVR. Jia and Yang (2021) compared deep neural networks and LSTMs against GARCH and ARCH models. Although the LSTMs outperformed the ARCH/GARCH models, the difference was not significant, and care

was not taken to find the optimal hyperparameters of the ARCH/GARCH models. In both these approaches, the authors use only the series of financial returns as input to the networks.

A far more successful class of algorithms combines statistical methods with RNN methods. Di-Giorgi et al. (2023) and Hu, Ni, and Wen (2020) propose LSTM and bidirectional LSTM (BiLSTM) models that use GARCH model forecasts as their inputs. In both cases, the authors found that the hybrid models outperformed traditional GARCH models by a significant margin. Di-Giorgi et al. (2023) also compare the performance of LSTMs and BiLSTMs with and without the GARCH forecasts. In their article, they give each of the models the price innovations of copper as well as a set of other explanatory variables and perform two-week-out copper price prediction. They found that the models that were given GARCH price forecasts outperformed the models that were not. It is also worth noting that, although hard to compare to other results because of the uncommon dataset, they achieve significantly improved performance, potentially because of the use of exogenous explanatory variables.

Another approach taken by Nguyen et al. (2022) introduces a novel architecture called a statistical recurrent stochastic volatility (SR-SV) model. Their motivation is to better capture long memory and nonlinear autodependence phenomena in volatility forecasting. In their article, they combine a statistical recurrent unit (SRU) with a traditional stochastic volatility model to predict the volatility, z_t, as shown in Equation 1. Here, β_1 is the nonlinear component term. An SRU is a type of RNN that allows a vector of summary statistics h_t to move through the network using a moving average. The equations that govern the SRU are summarized in Equations 3 to 5. Here, $\alpha_1 \ldots \alpha_i$ are the moving-average weights, and W_h, b_r, W_r, W_x, and b_φ are trainable model parameters. They evaluate their network on three different simulated volatility datasets. Datasets 2 and 3 incorporate long memory and nonlinear autodependence into their simulation models. Although the first SR-SV model has comparable performance to SV modeling, it outperforms in the second two volatility models.

$$z_t = \beta_0 + \beta_1 \mathrm{SRU}(\eta_{t-1}, z_{t-1}, h_{t-1}) + \phi z_{t-1} + \epsilon_t^{\eta} \tag{1}$$

$$\eta_t = \beta_0 + \beta_1 h_t + \epsilon_t^{\eta} \tag{2}$$

$$r_t = \Psi(W_h h_t - 1 + b_r) \tag{3}$$

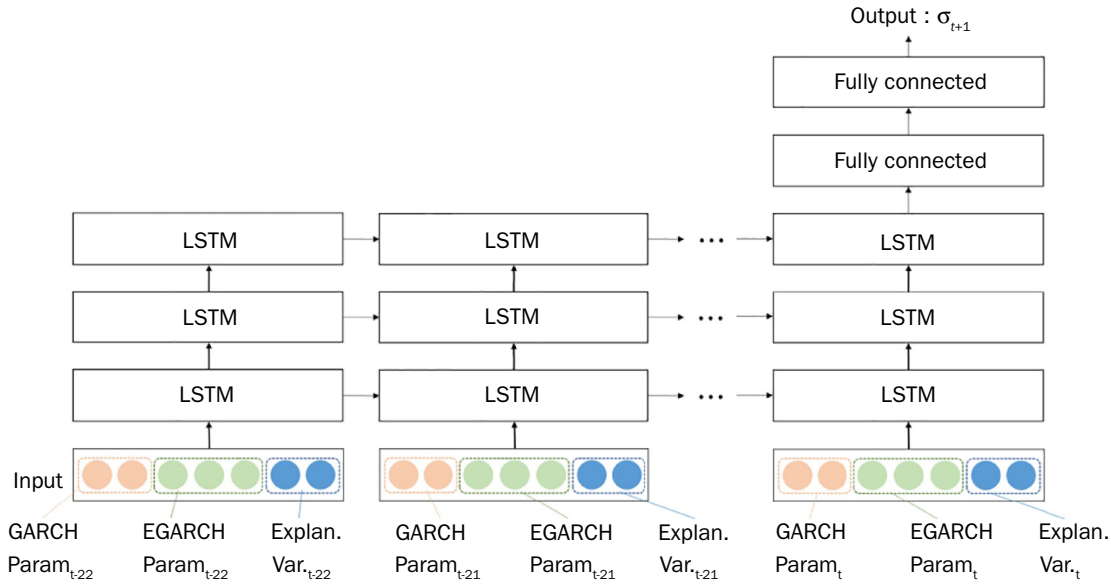
$$\varphi_t = \Psi(W_r r_t + W_x x_t + b_\varphi) \tag{4}$$

$$h_t^{(\alpha_j)} = \alpha_j h_{t-1}^{(\alpha_j)} + (1 - \alpha_j)\varphi_t, j = 1, \ldots, m; h_t = (h_t^{(\alpha_1)}, \ldots, h_t^{(\alpha_m)})^{\top} \tag{5}$$

One of the issues with RNNs is that they need a lot of training data before they begin to outperform other methods. Many of the previous methods assume fixed parameters, and so the model may not generalize well to all financial assets. Kim and Won (2018) propose an LSTM that takes the parameters of traditional statistical techniques, such as GARCH and EGARCH, as one of its inputs. These parameters are estimated using maximum likelihood estimation (MLE), and the model can now be trained on time series with many different parameters. They also included additional explanatory variables such as the price of gold and oil, the the 3-year AA-grade corporate bond (CB) interest rate, and the the 3-year Korea Treasury Bond (KTB) interest rate. A diagram of this model is shown in Exhibit 1. In the study, they used their model to perform one-day-out volatility predictions for the KOPSI 200 Index. They concluded that neural networks have two advantages in volatility prediction. The first is that they

EXHIBIT 1

Diagram of Kim and Won's Hybrid LSTM Model from Kim and Won (2018)



are able to mitigate the weaknesses of different GARCH-type models. The second is that they are able to utilize econometric information such as gold prices to improve prediction, whereas in standard models these are assumed to be stationary.

Convolutional neural networks (CNNs). CNNs are most often used for image-related tasks because of their spatial pattern recognition capabilities (Krizhevsky, Sutskever, and Hinton 2012; Goodfellow, Bengio, and Courville 2016). Although RNNs are more commonly used because of their superior sequential modeling capabilities in time-series forecasting tasks, there has been some work in the field using CNNs for volatility forecasting.

Doering, Fairbank, and Markose (2017) used CNNs to predict stock price and price volatility using data from the London stock exchange. Instead of using just the price as an input, they used matrix representations of the order book, trades, orders, and deletions (of bids/asks) of each stock. In particular, each column in the matrix represents a particular time window, and each row represents a price point. In their results, it can be seen that CNN was able to extract meaningful features and perform successful forecasting, likely because of the additional information (Exhibit 2). Interestingly, the model performed significantly better in the task of volatility prediction compared to the task of price prediction.

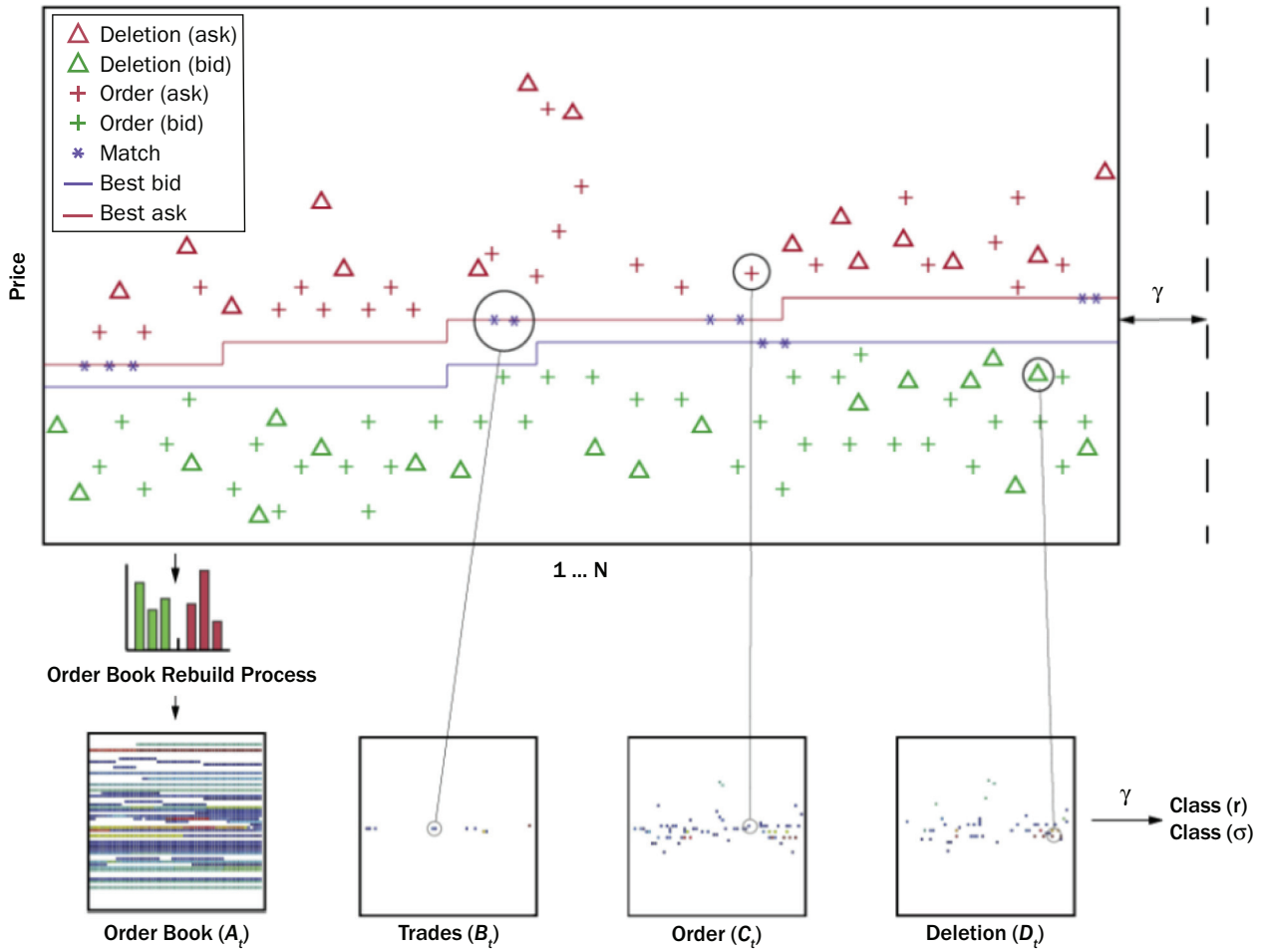
Particle Filtering

Particle filtering for parameter estimation. Monte Carlo methods have long been employed for the parameter estimation of stochastic volatility models via MLE (Sandmann and Koopman 1998). Malik and Pitt (2009) showed that a PF could also be employed to perform likelihood inference on stochastic volatility models and therefore optimize parameters. The aim is to estimate the likelihood:

$$\log L(\theta) = \log f(y_{1,\ldots,}y_T|\theta) = \sum_{t=1}^{T} log f(y_{t+1}|\theta;Y_t) \tag{6}$$

EXHIBIT 2

Diagram Showing How Options Data Are Converted to the Inputs for the CNN in Doering, Fairbank, and Markose (2017)



We can approximate $f(y_{t+1}|\theta; Y_t)$ by taking our previous samples/particles from $f(h_t|Y_t; \theta)$, sampling from the transition density function $f(h_{t+1}|h_t; \theta)$, and then exploiting the following relationship:

$$f(y_{t+1}|\theta; Y_t) = \int f(y_{t+1}|h_{t+1}; \theta) f(h_{t+1}|Y_t; \theta) dh_{t+1} \tag{7}$$

Particle filtering for volatility estimation. PFs have also been used extensively to directly estimate volatility using online filtering. The most common method for building a simple sequential importance resampling (SIR) PF implementation, also known as a bootstrap filter, for a particular stochastic volatility model, requires two things:

1. The definition of a transition density function: This can be derived from the discretized version of the volatility dynamics of the stochastic volatility model in question.
2. The definition of an observation likelihood function: This is derived from the equations relating the estimated volatility to the size of the returns; normally a noise parameter is used at this stage, the shape of which heavily determines the shape of the observation likelihood function.

Malik, Pitt, and Doucet build PFs for four stochastic volatility models: SV, stochastic volatility with leverage (SVL), stochastic volatility with leverage and jumps (SVLJ), and SV-GARCH (Pitt, Malik, and Doucet 2014). They show their derivations of these two functions, including their newly introduced SV-GARCH model. They then evaluate the models on real-world data, using log likelihood as their main criterion. Pitt and Shephard observed that PFs suffer from two main issues. First, when there is an outlier, the weight distribution is uneven and so a large number of particles is required to draw a distribution close to the empirical sampling density. Second, the tails of the distribution $p(y_t|x_t)$ are often poorly approximated because of a lack of particles. To solve this, they implement the auxiliary PF (APF) (Pitt and Shephard 1999). Their key innovation was to introduce an auxiliary variable k, which is then used to draw samples from $x^j_{t+1}, k^j \sim g(x_{t+1}, k|Y_{t+1})$. Afterward, the weights are readjusted using the following equation:

$$w^j_{t+1} = \frac{f(y_{t+1}|x^j_{t+1})f(x^j_{t+1}|x^{k^j}_t)}{g(x^j_{t+1}, k^j|Y_{t+1})} w^j_t \tag{8}$$

$g(.)$ can be designed to make the weights more even. In their paper, Malik and Pitt apply this general algorithm to both an ARCH model and a stochastic volatility model. Song, Liang, and Liu (2014) also apply the APF to model stochastic volatility with jumps. They observed that, because the tail ends of the observation function are poorly estimated by an SIR PF, jumps in volatility are often poorly approximated. To solve this, they designed their transition function so that it produces $k_\lambda = \max\{1, [\lambda \cdot m]\}$ particles with jumps, where λ is the probability of a jump and m is the total number of particles. The effect of this change can be seen in the diagram from their article, shown in Exhibit 3. Although the APF does not perfectly model the distribution, there are at least a good number of particles within the high probability region of the postjump distribution.

PF-RNNS

Although PFs are a good algorithm for approximating the current state in a nonlinear state space, care is needed to create good observation likelihood and state transition functions. Karkus, Hsu, and Lee (2018) introduced the PF network (PF-Net), an end-to-end differentiable implementation of the PF (Exhibit 4). The PF-Net was then used in a two-dimensional localization and visual odometry task. Their work was then further extended by Ma, Karkus, and Hsu (2020), who improved the basic PF-Net by implementing it as an LSTM and gated recurrent unit (GRU). They used the network for several tasks but focused on localization within a small generated maze.

Intuitively, a PF-Net or PF-RNN is a type of recurrent neural network that has not just one hidden state h_t but also K hidden belief states with an associated weight $H_t = \{(h^i_t, w^i_t)\}^K_{i=0}$. These belief states can be thought of as the particles. Unlike particles, however, they do not necessarily need to directly represent the system's state but can be a latent representation of it. In a regular RNN, the hidden state goes through a deterministic update; however, in a PF-RNN, they instead go through a stochastic Bayesian update $h^i_t = f_{trans}(h^i_{t-1}, u_t, \xi^i_t)$, where u_t is the control vector and $\xi^i_t i$ is a noise term. The weights are then updated using a learned observation likelihood function $w^i_t = f_{obs}(x_t, h^i_t)w^i_{t-1}$, where x_t is the observation. In both articles, these functions are represented as neural networks with learnable parameters.

One of the issues with building a PF-RNN is that resampling is not differentiable. To fix this, Karkus, Hsu, and Lee (2018) introduce *soft resampling*. Instead of sampling from the real distribution $p(i) = w^i_t$, we sample from another distribution

EXHIBIT 3

Diagram from Song, Liang, and Liu (2014) Showing the Difference in the Estimation Capabilities of the Vanilla Particle Filter and the Auxiliary Particle Filter

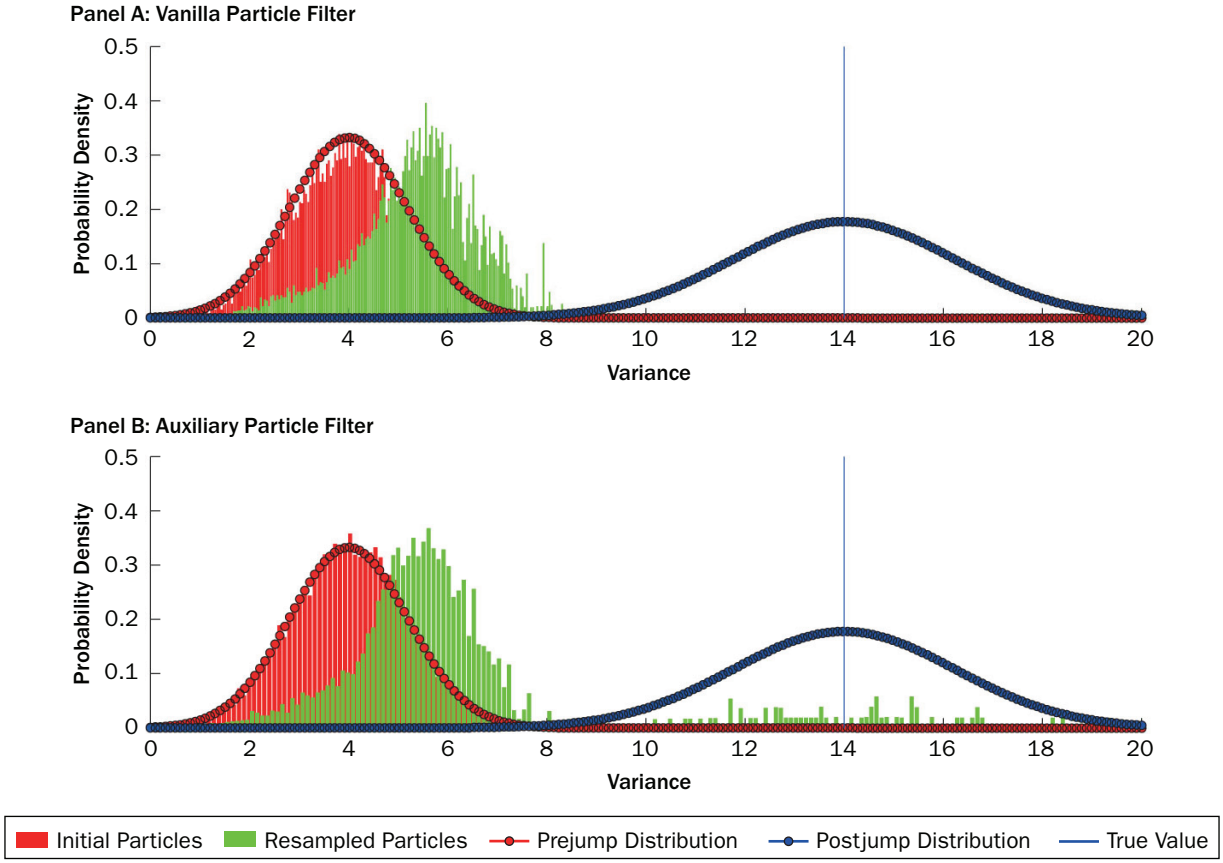
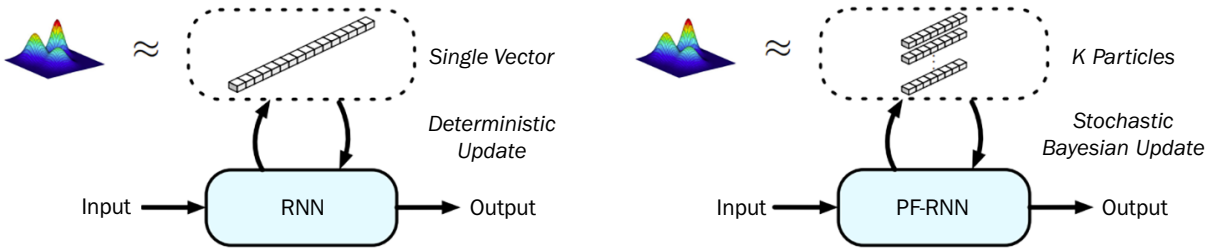


EXHIBIT 4

Diagram of the Basic Difference between an RNN and a PF-RNN from Ma, Karkus, and Hsu (2020)



$q(i) = \alpha p(i) + (1-\alpha)\left(\frac{1}{K}\right)$. This is a combination of the desired distribution $p(i)$ and a normal distribution and gives nonzero gradients when α is larger than 0. The new weights are then calculated using the importance sampling formula:

$$w_t'^i = \frac{p(k)}{q(k)} \tag{9}$$

Ma, Karkus, and Hsu (2020) also introduce a new loss function. They note that the goal of a PF is not just to provide an accurate estimate of the exact state but also to approximate the true probability distribution of the belief state. To do this, they optimize an evidence lower bound (ELBO) loss that uses sampled particles:

$$L_{\text{ELBO}}(\theta) = -\sum_{t \in \mathcal{O}} log \frac{1}{K} \sum_{i=1}^{K} p(y_t | \tau_{1:t}^i, x_{1:t}, \theta) \tag{10}$$

where $\tau_{1:t}^i$ gives the history of samples chosen and random noise inputted into a particle. The value of $p(y_t|\tau_{1:t}^i, x_{1:t}, \theta)$ is approximated by assuming the distribution is a Gaussian with mean 0 and variance 1. They combine this loss with traditional mean square error loss, weighting it with the parameter β:

$$L(\theta) = L_{MSE}(\theta) + \beta L_{ELBO}(\theta) \tag{11}$$

CONTRIBUTION

Overview

Our goal is to predict the volatility of a financial asset over time using a series of observations of its underlying price movements. We can assume we have corresponding pairs of sequences to use as training data. In particular, we have a sequence of observations $X_1 \ldots X_t$ and the sequence of corresponding unobservable volatilities $Y_1 \ldots Y_t$, and our goal is to estimate the volatility at each time step $\hat{Y}_1 \ldots \hat{Y}_t$.

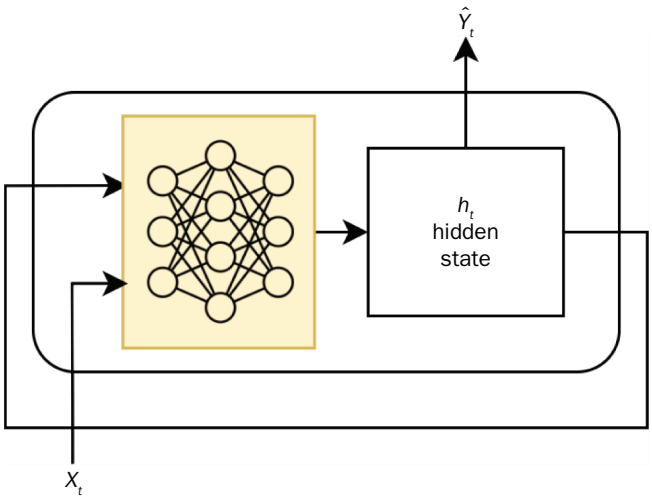
An approach using an RNN would involve creating a network that takes the price change X_t and a previous hidden state h_{t-1} and then outputs the predicted volatility Y_t and the updated hidden state h_t. The hidden state h_t and the predicted volatility Y_t do not necessarily need to be the same but can be depending on the approach. The network could then be trained using the sequences of data. A diagram of this approach is shown in Exhibit 5.

A PF approach requires the definition of a state transition equation $f_{trans}(p)$, where p is a particle representing the potential state, and the observation likelihood function $f_{obs}(p, x)$, which, given a price movement observation x, returns the likelihood of that particle. The particle's state could just be the predicted volatility, or we can store more information such as the previous n volatilities if we want to use a model such as a GARCH model. The observation could also be generalized to the previous m observations for the same reason. Our transition equation and likelihood function would be based on one of the volatility models we have written about in our background section.

One idea is to adapt the PF-RNN structure to the task of volatility forecasting. We can extend the RNN so that, instead of a single hidden state h_t, it has several hidden states, $\{p_t^i\}_{i=0}^n$ that each represent a belief state, whether that's just the current volatility estimate or some hidden representation. We can think of these hidden states as particles, and along with the belief state we store the weight associated with each particle, so we have $\{(p_t^i, w_t^i)\}_{i=0}^n$. The transition function and

EXHIBIT 5

Standard RNN Implementation



observation likelihood function are now represented by neural networks that we can train. At each stage the RNN first updates each of the particles and then uses the current observation and new set of particles to update each of the particle's weights. Finally, the particles are resampled using a differentiable form of resampling called soft resampling. We can take the mean of this new set of particles as the observation.

This is the idea behind the basic SV-PF-RNN that we will be developing in this section (Exhibit 6). We introduce new architecture to better predict stochastic volatility. We also add randomness so that the transition function can be fully approximated. To help the model reach similar performance to a PF, we pretrain its networks to approximate a PF. We then make several modifications to the loss function to help train the network despite the inherent randomness of volatility calculation and the issues this creates with low-probability outcomes in an SV-PF-RNN. Finally, we train our network to show that it can outperform the standard PF.

Taylor SV-PF-RNN

The Taylor stochastic volatility model. To evaluate the performance of our SV-PF-RNN, we will generate data using the relatively simple stochastic volatility model introduced by Taylor (1987). The equations for the innovation and the volatility prediction are

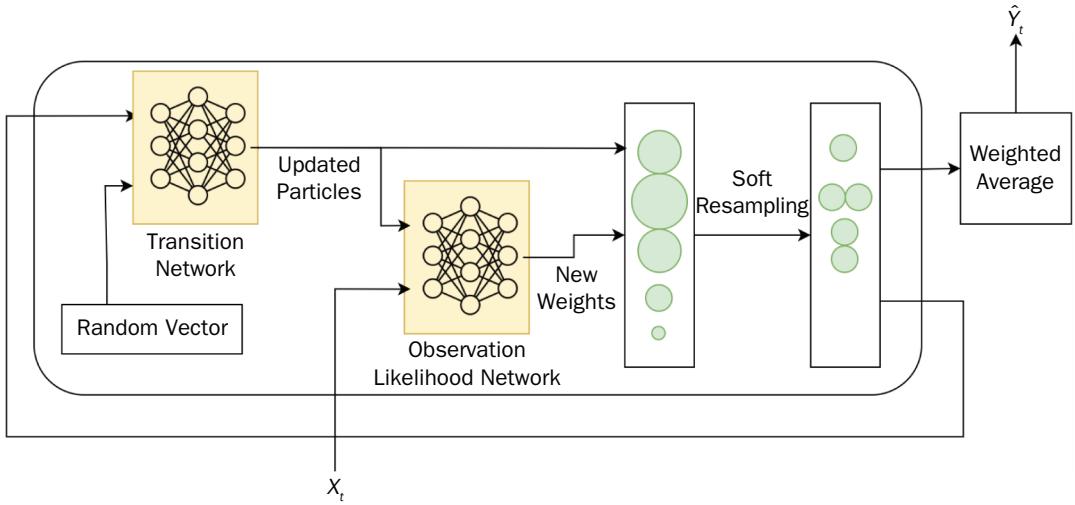
$$\sigma_t = \mu + \phi(\sigma_{t-1} - \mu) + \zeta_\sigma \tag{12}$$

$$x_t = \sigma^2 \zeta_x \tag{13}$$

where $\zeta_\sigma \sim \mathcal{N}(0, \tau)$ and $\zeta_x \sim \mathcal{N}(0, 1)$. Note that the model takes three parameters: μ, ϕ, and τ. We will assume these are fixed. We chose to start with this model because it is simple and will allow us to see if our SV-PF-RNN is able to estimate basic stochastic volatility.

PF implementation. Based on this, we can define the state transition equation and observation likelihood function for a PF implementation. ε is a noise variable where $\varepsilon \sim \mathcal{N}(0, \tau)$. Note that the parameters of the volatility generation are fixed and known, so the state consists only of the predicted volatility.

EXHIBIT 6
Diagram of Our SV-PF-RNN



NOTE: We give credit to David McDonald from The Noun Project for the neural network icon.

$$f_{obs}(p,x)=\frac{1}{\sqrt{2\pi p^2}}e^{-\frac{x^2}{2p^2}} \tag{14}$$

$$f_{trans}(p,\varepsilon)=\mu+\phi(p-\mu)+\varepsilon \tag{15}$$

Overview of the SV-PF-RNN. Our SV-PF-RNN maintains a set of n particles and associated weights, $\{(p_t^i, w_t^i)\}_{i=0}^n$, where each particle is a single scalar representing the estimated volatility at that point in time. At each time step we input a single observation X_t, then update this belief state using the observation and the previous belief state, and finally output a single estimate of the volatility for that time step, $\hat{Y}_t$, based on the belief state.

In our SV-PF-RNN, we use two neural networks to represent our transition function and observation likelihood function. We replicate our transition function, $p_t = f_{trans}(p_{t-1})$, which can be approximated with a neural network. One issue with this is that the original transition function contains the process noise variable ε. In Ma, Karkus, and Hsu (2020), the authors add randomness after the transition function, $p_t = f_{trans}(p_{t-1}) + \varepsilon$, where ε has a fixed variance and mean. In several volatility models, however, the noise is nonlinearly used in the transition stage. To fix this, we include this noise as an input, and so our transition equation is represented by $p_t = f_{trans}(p_{t-1}, \varepsilon)$, where ε is a vector of random numbers drawn from a normal distribution with variance 1 and mean 0. We will discuss the implications of including a random input in the training process later.

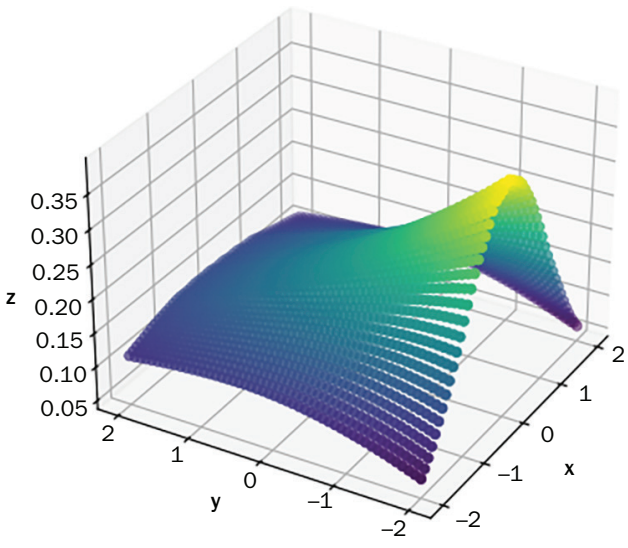
Our observation likelihood function is also represented by a neural network that takes as input the particle and the price observation and outputs the likelihood, $w_t = f_{obs}(p_t, x_t)$. Finally, after updating and normalizing all our new weights, we then resample the particles using soft resampling. To produce our estimated volatility $\hat{Y}_t$, we simply take the average of all of the particles. Because we resampled in the previous step, there is no need to weight each estimate.

Pretraining the networks. To speed up the training process, we first train the network to replicate our PF implementation. Neural networks can be thought of as nonlinear function approximators (Hornik, Stinchcombe, and White 1989). Hence, we can individually train the observation likelihood and transition function networks to approximate their equivalent functions in the PF implementation. We can do this by generating input–output data pairs for each of these functions and training the networks on this. We then load the weights into the full model. We have plotted what this looks like for our neural network in Exhibits 7 and 8.

In our evaluation, we show that the performance of this PF approximator network is similar to that of the basic PF. While training, however, we only produced input data without a certain numerical range. In practice, values outside this range can sometimes be found, and so the network's behavior for these values can be unstable.

EXHIBIT 7

Observation Likelihood Function



Loss function. One of the major issues that we encountered after training the network for a long time is that the resultant models would just output a constant value close to the mean of all sequences. Upon visual inspection, we found that all the particles were converging to this value, as can be seen in Exhibit 9. This means that either the transition function is flattening and converting all pairs of input to the mean value or that the observation likelihood function is giving low-probability scores to every value except the mean of all the sequences.

After further investigation we realized that because we have random input and because we're using gradient descent, certain behaviors such as tending to the mean are encouraged. This can be illustrated by the following example: Imagine we have some stochastic process governed by a random walk, which centers around a mean of 0. Our goal is to train the neural network so that it models this equation in the transition function, given the input of the previous volatility and some random input. Now let's imagine we have a particle with a belief state of 0.5. Assuming our neural network is correctly pretrained, the particle is equally as likely to move to 1 as it is to 0. Because the stochastic process is centered around 0, the particle is likely to be penalized less if it moves to 0. Slowly, the network learns not the transition function but rather that, regardless of the random input, it should move the particle to the mean position (i.e., the position its least likely to be penalized in).

This results in the model very easily falling into a local minima in which the transition function simply transforms particles toward the mean. Alternatively, the model could simply learn an observation probability function that assigns very high weights to particles close to the mean, thereby causing particle depletion everywhere else. To solve this, we introduce a new loss function that weights the loss on each particle by the inverse likelihood of the belief state being at that position (given no prior information):

$$L(\theta) = \sum_{t=0}^{T} \sum_{i=0}^{N} \frac{L(\theta \mid p_t^i)}{f(p_{t+1}^i)} \tag{16}$$

EXHIBIT 8

Transition Function

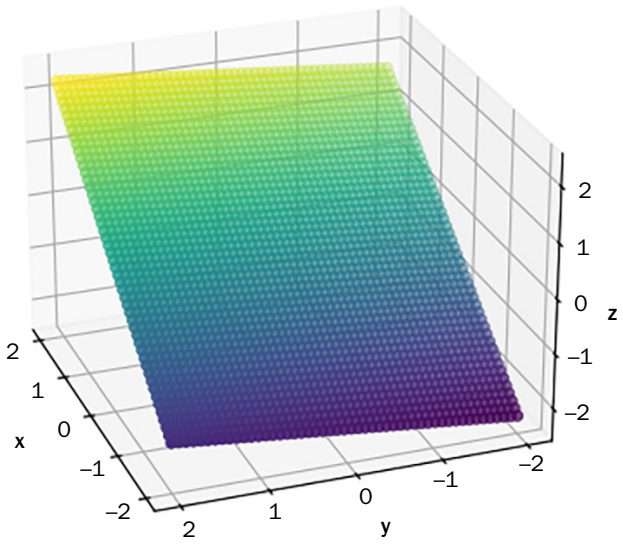
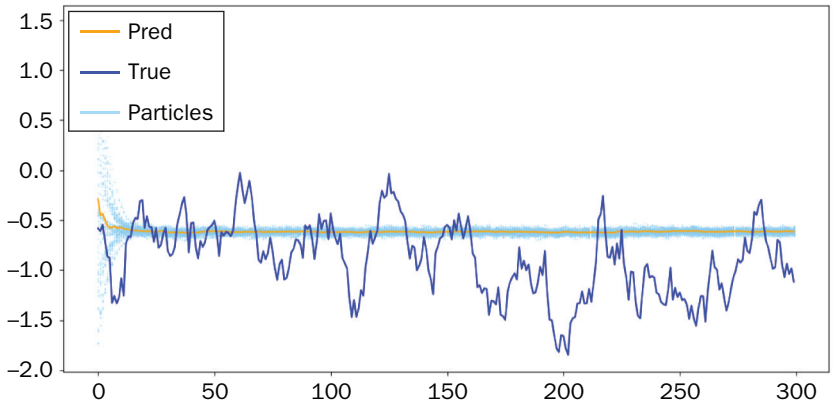


EXHIBIT 9

Mean Convergence Issue after Training



NOTE: Despite the particles starting with a large spread, they soon all converge to a value of around −0.5.

In the case that our volatility model calculating $f(p_{t+1}^{i})$ is intractable, we use a Gaussian distribution, with variance τ and the mean μ as an approximation (Exhibit 10).

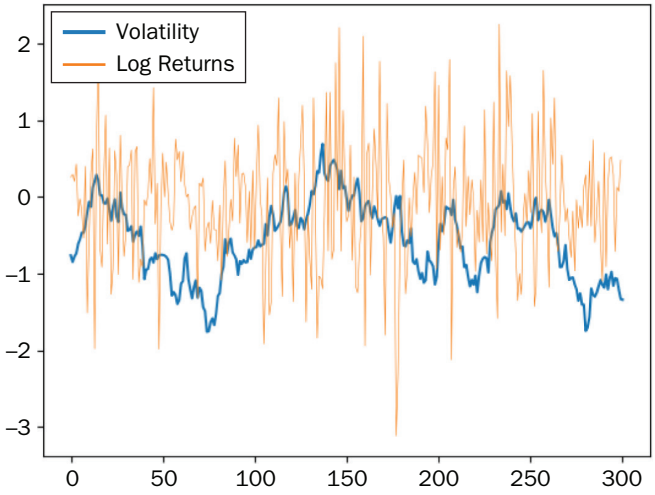
Compute unified device architecture (CUDA) compatibility. In order to train the model faster, the code was Cudified (NVIDIA, Vingelmann, and Fitzek 2020). CUDA is a framework that makes it easier to accelerate computation by enabling it to be run on parallel computing devices. In the case of our SV-PF-RNN training, the neural networks can be massively sped up by training on a GPU. At first, the size of the data, because of the length of the series and number of particles, caused the GPU to keep running out of memory. So, we reduced the batch size to 50 and modified the code to clear the CUDA cache before running. Another side effect of running on a machine with a GPU is that random numbers may be generated slightly differently than they would be on a CPU. Analyzing the effects of this was outside the scope of this work.

RESULTS AND EVALUATION

In this section, we evaluate the performance of our SV-PF-RNN, comparing it to the performance of a PF. We first go over the details of our experiment, and the metrics we use for comparison. We then present the results of these experiments (Exhibits 11–13). Finally, we evaluate the performance of our PF, including an analysis of its performance with outliers and performance with different numbers of particles.

EXHIBIT 10

Example of Simulated Volatility along with the Simulated Price Innovations



Description of the Experiment

Evaluation metrics. For our evaluation, we use MSE, mean absolute error (MAE), Quasi-Likelihood (QLIKE), mean directional accuracy (MDA), and log likelihood:

$$MSE = \frac{1}{n}\sum_{t=0}^{T}(y_t - \hat{y}_t)^2 \tag{17}$$

$$MAE = \frac{1}{n}\sum_{t=0}^{T}|y_t - \hat{y}_t| \tag{18}$$

EXHIBIT 11

Mean Results from the Experiment Described in 5.2

Data Generated Using Taylor SV with (μ = −0.6558, τ = 0.1489, ϕ = 0.9807)				
Model Type	Particle Filter	SV-PF-RNN (pretraining only)	SV-PF-RNN (standard loss function)	SV-PF-RNN (modified loss function)
MSE	0.2210	0.2055	0.1823	**0.1652**
MAE	0.3734	0.3637	0.3420	**0.3246**
QLIKE	0.07510	–1.7669	–1.2038	–0.8149
MDA	0.4970	0.4975	0.4946	0.4970
Log Likelihood	–821	–1094	**–739**	–822

NOTE: The best results are highlighted in boldface (if significant).

EXHIBIT 12

Observation Likelihood Function in Trained Model

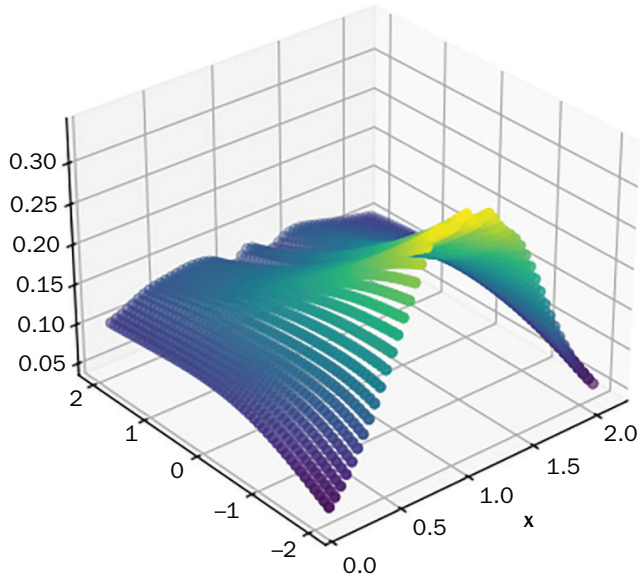
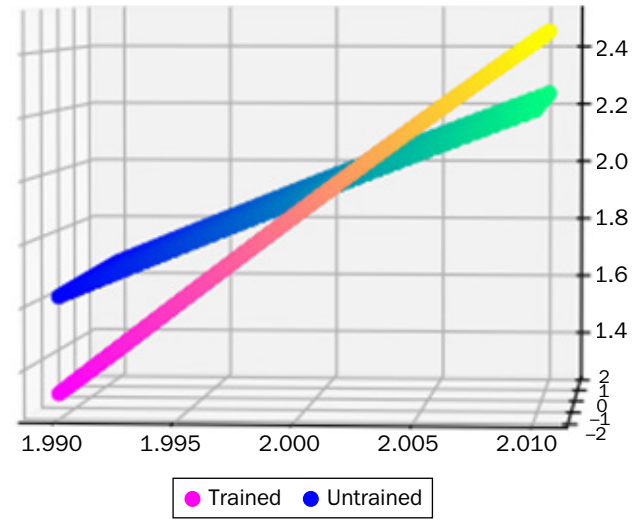


EXHIBIT 13

Transition Functions from Trained and Untrained Models



$$QLIKE = \frac{1}{n}\sum_{t=0}^{T}\left(log(\hat{y}_t^2) - \frac{y_t^2}{\hat{y}_t^2}\right) \tag{19}$$

$$MDA = \frac{1}{n}\sum_{t=1}^{T} 1_{sign(\hat{y}_t - \hat{y}_{t-1}) == sign(y_t - y_{t-1})} \tag{20}$$

$$LogLikelihood = \sum_{t=0}^{T} log\left(\frac{1}{\sqrt{2\pi\tau^2}} e^{-\frac{(\hat{y}_t - y_t)^2}{2\tau^2}}\right) \tag{21}$$

Experimental setup. For our experiment we generate $K = 3{,}000$ random paths of length $T = 300$ using the stochastic volatility model described earlier. We fix our parameters at ($\mu = -0.6558$, $\tau = 0.1489$, $\varphi = 0.9807$). For the SV-PF-RNN, 2,000 of these paths are used to train the network, 500 are used for testing, and 500 are used for evaluation.

RESULTS

Evaluation and discussion. After evaluating each of our models, we find that our SV-PF-RNN with the modified loss function has the best performance out of all the models, followed by the SV-PF-RNN with the standard loss function. Interestingly, the untrained SV-PF-RNN has a slightly lower MSE than the standard PF, even though it is an approximation of the PF. We hypothesize this is because of the different resampling schema of the SV-PF-RNN, which prevents particles from depleting as quickly in low-likelihood areas. Another interesting observation is that the log likelihood of the PF is higher than the untrained SV-PF-RNN and around the same as the SV-PF-RNN with the modified loss function. The exact shape of the log likelihood graph depends on the parameter τ. With $\tau = 0.1489$, log likelihood penalizes distant outliers more than MSE, suggesting that the models with lower MSE may still suffer from having more distant outliers.

By plotting the particles and estimated volatility for our untrained and trained SV-PF-RNNs—Exhibits 14 and 15, respectively—we can visually see the difference between the two models. We can see that the SV-PF-RNN is a lot more responsive to sudden and large changes in volatility. Additionally, the spread of particles is far larger.

Interpretability. Another benefit of our SV-PF-RNN is improved interpretability. By plotting the observation likelihood and transition functions, we can see how they've changed to get some insight into how the model has improved predictions. For real-world data, this could also give an insight into the estimated dynamics of a real-world system. In the following, we have produced these two plots. We can see that

EXHIBIT 14

Pretraining Only

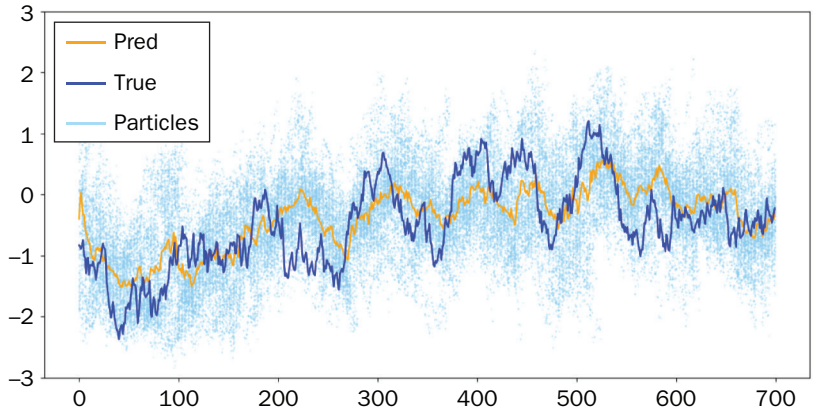


EXHIBIT 15

Fully Trained Model

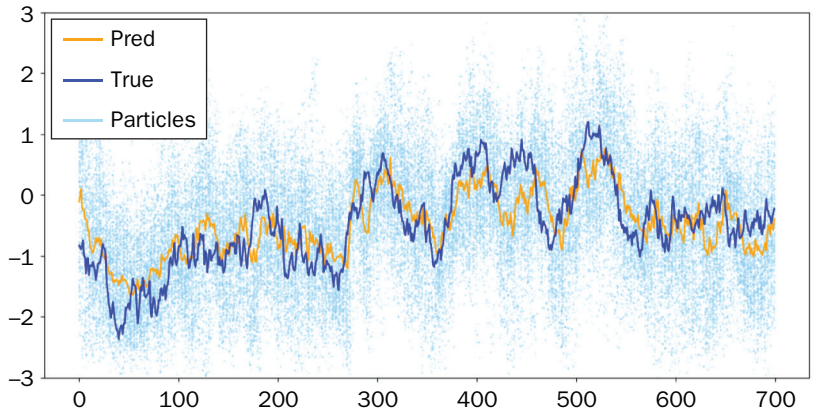


EXHIBIT 16

Performance of the Three Models on Outliers

Data Generated Using Taylor SV with ($\mu = -0.6558$, $\tau = 0.1489$, $\phi = 0.9807$)			
Model Type	**Particle Filter**	**SV-PF-RNN (pretraining only)**	**SV-PF-RNN (fully trained)**
MSE	**0.3313**	0.5452	0.4654
MAE	**0.4527**	0.5782	0.5363

NOTE: The best results are highlighted in boldface.

the observation likelihood has scored particles right next to the mean as lower than in the original (plotted in the contribution section). We can see the transition function has become a lot steeper, meaning that the spread of particles after the transition will be far larger. This is consistent with the larger particle spread seen in the graphs. Because we are generating data, we know that these are not the true observation likelihood and transition functions but rather may just improve the model's performance as an estimator according to our loss function.

Performance on outliers. One of the disadvantages of machine learning methods over statistical methods is that they often perform badly with outliers in the data (Exhibit 16). To test our method against a regular PF, we created a dataset in which the minimum or maximum values in each of the volatility series was in the 3rd or 97th percentile of all minimum or maximum values, respectively. We then calculated the same set of statistics and plotted examples of the results.

EXHIBIT 17
Particle Filter

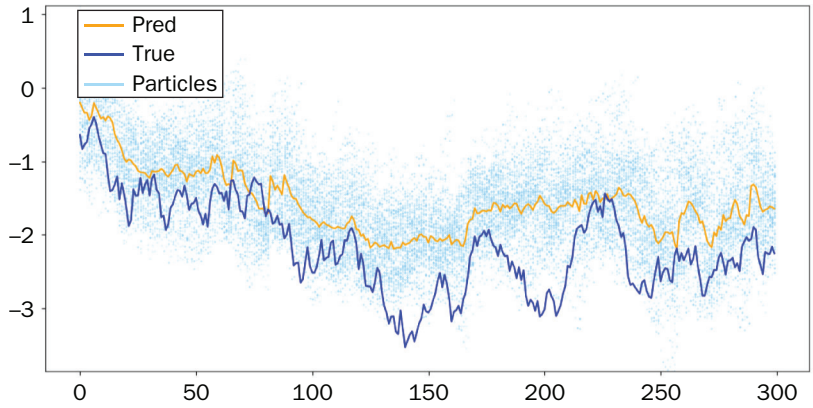
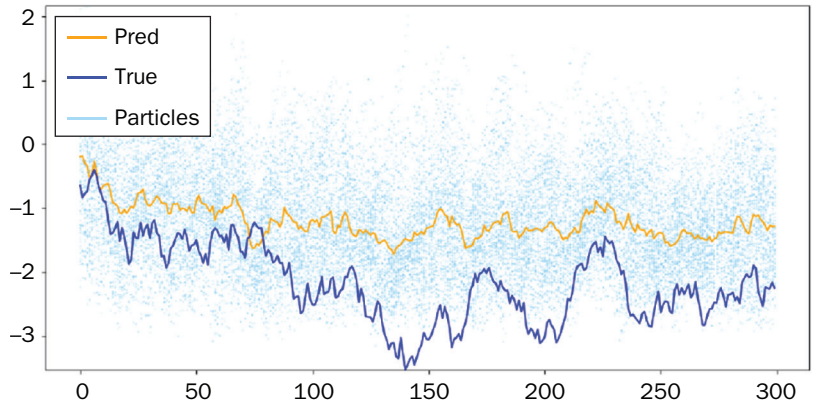


EXHIBIT 18
Fully Trained Model



The model does not perform as well as the PF, with a 40.4% increase in the MSE and an 18.5% increase in the MAE. The increase in the MSE is far more significant than the increase in the MAE, suggesting that the PF-RNN has far more regions in which it is significantly off from the true value of volatility. We can see in Exhibits 17 and 18 that it often fails to converge to the true value of volatility for long periods of time.

Initially, we hypothesized that is probably due to the fact that there were not as many extreme examples in the training data, and so the model fails to generalize to these extreme cases. The untrained model performs even worse than the trained model, which suggests that the pretrained neural networks are simply not able to handle a large enough data range. There are two potential solutions to this problem:

1. Pretrain the network on a wider range of inputs (this may also require using larger networks).
2. Balance the training dataset to include more examples of series with outliers.

It is also worth noting the work of Pitt and Shephard (1999), who observed that PFs often poorly approximate the true tails of $p(x_t|y_t)$, where x_t is the true state and y_t an observation. This appears to be a weakness that our PF does not overcome.

In their article, they adopt the APF to mitigate the issues of particle degeneracy. By adapting our SV-PF-RNN to include auxiliary variables that encourage particle diversity, we may also be able to improve performance on outliers. It should also be noted that this is not necessarily an issue with our SV-PF-RNN compared to the PF as much as it is an issue with the type of PF we have used.

CONCLUSION

In this study, we proposed a novel hybrid architecture that combines a PF with an RNN for volatility forecasting. Our objective was to enhance the performance of a PF and assess its effectiveness in both generated data and real-world scenarios.

First, we successfully improved the performance of a PF on the task of estimating volatility from generated data. Through extensive experimentation and evaluation, we observed notable enhancements in our hybrid model's accuracy and responsiveness to changes in volatility. This improvement signifies the potential of our approach to outperform traditional PFs in capturing and forecasting volatility patterns.

Furthermore, we visually analyzed the generated data and plotted various graphs to investigate the behavior of our model. These visualizations demonstrated the increased responsiveness of our hybrid model to changes in volatility, reinforcing its ability to capture and adapt to dynamic market conditions.

When we applied our model to real-world data, we encountered challenges and achieved limited success. The nature of the real-world data differed significantly from the generated data used in our training process. The complex dynamics and unique characteristics of real-world volatility posed difficulties for our hybrid architecture, hindering its performance on this particular dataset.

Future Work

Despite the limitations encountered when training on real-world data, our study provides valuable insights into the capabilities and potential of hybrid PF-RNN models for volatility forecasting. Our findings suggest that further refinement and adaptation of our approach may be necessary to effectively tackle the intricacies and nuances present in real-world financial data.

Incorporating exogenous variables. One notable advantage of employing machine learning methods is their capacity to uncover intricate nonlinear patterns and relationships within data. By incorporating additional inputs after the pretraining phase, we can introduce exogenous explanatory variables that have the potential to enhance the model's ability to predict large volatility jumps. Previous research has demonstrated the effectiveness of including variables such as inflation rates or implied volatilities of related assets, leading to significant improvements in performance using machine learning techniques (Kim and Won 2018; Hu, Ni, and Wen 2020).

Adaptive filtering. A current limitation of our model is its assumption of static parameters governing volatility dynamics, despite evidence that parameters can change because of structural change in the market (Liu and Maheu 2008). The field of adaptive filtering offers promising avenues for improvement, involving continuous estimation of parameter values alongside volatility estimation (Liu and West 2001). An alternative implementation of our model could involve incorporating the estimated parameter values as latent variables, with each particle having its own estimates of each parameter. A separate neural network could then be employed to iteratively update and adapt these values in real time, resulting in a more dynamic and adaptive volatility forecasting framework.

Implementing the APF. One of the issues with our SV-PF-RNN was that it poorly estimated the tail ends of the distribution and thus failed to fit to outliers in the data. The APF has been proposed as a solution to this problem (Pitt and Shephard 1999). Implementing this into our SV-PF-RNN, thereby creating the SV-APF-RNN, would have some benefits over a normal APF if successful. Usually implementing the APF would require the careful design of the function $g(.)$. By implementing it as a differentiable network, the model can learn what this function should be.

REFERENCES

Ball, C. A., and A. Roma. 1994. "Stochastic Volatility Option Pricing." *The Journal of Financial and Quantitative Analysis* 29 (4): 589.

Black, F., and M. Scholes. 1973. "The Pricing of Options and Corporate Liabilities." *Journal of Political Economy* 81 (3): 637–654.

Bollerslev, T. 1986. "Generalized Autoregressive Conditional Heteroskedasticity." *Journal of Econometrics* 31 (3): 307–327.

Breiman, L. 2001. "Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author)." *Statistical Science* 16 (3): 199–231.

Burns, A., and W. Mitchell. 1946. *Measuring Business Cycles*. New York: National Bureau of Economic Research.

Chan, J. C. C., and A. L. Grant. 2016. "Modeling Energy Price Dynamics: GARCH versus Stochastic Volatility." *Energy Economics* 54: 182–189.

Christensen, K., M. Siggaard, and B. Veliyev. 2023. "A Machine Learning Approach to Volatility Forecasting." *Journal of Financial Econometrics* 21 (5): 1680–1727. https://doi.org/10.1093/jjfinec/nbac020.

Contributors. "Reproducibility." Copyright 2023, PyTorch Contributors. https://pytorch.org/docs/stable/notes/randomness.html.

Di-Giorgi, G., R. Salas, R. Avaria, C. Ubal, H. Rosas, and R. Torres. 2023. "Volatility forecasting Using Deep Recurrent Neural Networks as GARCH Models." *Computational Statistics*. https://link.springer.com/article/10.1007/s00180-023-01349-1#citeas.

Doering, J., M. Fairbank, and S. Markose. 2017. "Convolutional Neural Networks Applied to High-Frequency Market Microstructure Forecasting." In *2017 9th Computer Science and Electronic Engineering (CEEC)*, IEEE, Colchester, UK, pp. 31–36.

Ge, W., P. Lalbakhsh, L. Isai, A. Lenskiy, and H. Suominen. 2022. "Neural Network–Based Financial Volatility Forecasting: A Systematic Review." *ACM Computing Surveys* 55 (1): 1–30.

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

Hansen, P. R., and A. Lunde. 2005. "A Forecast Comparison of Volatility Models: Does Anything Beat a GARCH(1, 1)?" *Journal of Applied Econometrics* 20 (7): 873–889.

Hornik, K., M. Stinchcombe, and H. White. 1989. "Multilayer Feedforward Networks Are Universal Approximators." *Neural Networks* 2 (5): 359–366.

Hu, Y., J. Ni, and L. Wen. 2020. "A Hybrid Deep Learning Approach by Integrating LSTM-ANN Networks with GARCH Model for Copper Price Volatility Prediction." *Physica A: Statistical Mechanics and Its Applications* 557: 124907.

Jia, F., and B. Yang. 2021. "Forecasting Volatility of Stock Index: Deep Learning Model with Likelihood-Based Loss Function." *Complexity* 2021: 1–13.

Karkus, P., D. Hsu, and W. S. Lee. 2018. "Particle Filter Networks with Application to Visual Localization." In *Proceedings of the 2nd Conference on Robot Learning*, edited by A. Billard, A. Dragan, J. Peters, and J. Morimoto, pp. 169–178. PMLR. https://proceedings.mlr.press/about/.

Kim, H. Y., and C. H. Won. 2018. "Forecasting the Volatility of Stock Price Index: A Hybrid Model Integrating LSTM with Multiple GARCH-Type Models." *Expert Systems with Applications* 103: 25–37.

Koopmans, T. C. 1947. "Measurement without Theory." *The Review of Economics and Statistics* 29 (3): 161.

Krizhevsky, A., I. Sutskever, and G. E. Hinton. 2012. "ImageNet Classification with Deep Convolutional Neural Networks." In *Advances in Neural Information Processing Systems*, vol. 25, edited by F. Pereira, C. J. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., Morehouse Lane, Red Hook, NY, USA.

Liu, C., and J. M. Maheu. 2008. "Are There Structural Breaks in Realized Volatility?" *Journal of Financial Econometrics* 6 (3): 326–360.

Liu, J., and M. West. 2001. "Combined Parameter and State Estimation in Simulation-Based Filtering." In *Sequential Monte Carlo Methods in Practice*, pp. 197–223. New York: Springer.

Liu, Y. 2019. "Novel Volatility Forecasting Using Deep Learning–Long Short Term Memory Recurrent Neural Networks." *Expert Systems with Applications* 132: 99–109.

Ma, X., P. Karkus, and D. Hsu. 2020. "Particle Filter Recurrent Neural Networks." *Proceedings of the AAAI Conference on Artificial Intelligence* 34: 5101–5108.

Makridakis, S., E. Spiliotis, and V. Assimakopoulos. 2018. "Statistical and Machine Learning Forecasting Methods: Concerns and Ways Forward." *PLOS ONE* 13 (3): e0194889.

Malik, S., and M. K. Pitt. 2009. "Modelling Stochastic Volatility with Leverage and Jumps: A Simulated Maximum Likelihood Approach via Particle Filtering." Working paper, University of Warwick.

——. 2011. "Modelling Stochastic Volatility with Leverage and Jumps: A Simulated Maximum Likelihood Approach via Particle Filtering." SSRN 1763783.

Nguyen, T.-N., M.-N. Tran, D. Gunawan, and R. Kohn. 2022. "A Statistical Recurrent Stochastic Volatility Model for Stock Markets." *Journal of Business Economic Statistics* 41 (2): 414–428.

NVIDIA, P. Vingelmann, and F. H. P. Fitzek. 2020. CUDA, release: 10.2.89.

Pitt, M. K., S. Malik, and A. Doucet. 2014. "Simulated Likelihood Inference for Stochastic Volatility Models Using Continuous Particle Filtering." *Annals of the Institute of Statistical Mathematics* 66 (3): 527–552.

Pitt, M. K., and N. Shephard. 1999. "Filtering via Simulation: Auxiliary Particle Filters." *Journal of the American Statistical Association* 94 (446): 590–599.

Sandmann, G., and S. J. Koopman. 1998. "Estimation of Stochastic Volatility Models via Monte Carlo Maximum Likelihood." *Journal of Econometrics* 87 (2): 271–301.

Shephard, N. 2005. "Stochastic Volatility." Economics Papers 2005-W17, Economics Group, Nuffield College, University of Oxford.

Shephard, N., and T. G. Andersen. 2009. "Stochastic Volatility: Origins and Overview." In *Handbook of Financial Time Series,* pp. 233–254. Berlin/Heidelberg: Springer.

Simonian, J. 2020. "Modular Machine Learning for Model Validation: An Application to the Fundamental Law of Active Management." *The Journal of Financial Data Science* 2 (2): 41–50.

Simonian, J., and F. J. Fabozzi. 2019. "Triumph of the Empiricists: The Birth of Financial Data Science." *The Journal of Financial Data Science* 1 (1): 10–13.

Simonian, J., and C. Wu. 2019. "Minsky vs. Machine: New Foundations for Quant-Macro Investing." *The Journal of Financial Data Science* 1 (2): 94–110.

Song, B., E. Liang, and B. Liu. 2014. "American Option Pricing Using Particle Filtering under Stochastic Volatility Correlated Jump Model." *Journal of Systems Science and Information* 2 (6): 505–519.

Taylor, S. J. 1987. "Forecasting the Volatility of Currency Exchange Rates." *International Journal of Forecasting* 3 (1): 159–170.

Tsay, R. S. 2010. *Analysis of Financial Time Series*, 3rd ed., p. 110. Hoboken, NJ: Wiley-Blackwell.

Zhang, G. P. 2003. "Time Series Forecasting Using a Hybrid ARIMA and Neural Network Model." *Neurocomputing* 50: 159–175.

Zivot, E. 2009. "Practical Issues in the Analysis of Univariate GARCH Models." In *Handbook of Financial Time Series,* pp. 113–155. Berlin/Heidelberg: Springer.