INSTITUTE
OF ECONOMIC STUDIES
Faculty of Social Sciences
Charles University

HISTORICAL CALIBRATION OF SVJD MODELS WITH DEEP LEARNING

Milan Fičura
Jiří Witzany

IES Working Paper 36/2023

Institute of Economic Studies,
Faculty of Social Sciences,
Charles University in Prague

[UK FSV – IES]

Opletalova 26
CZ-110 00, Prague
E-mail : ies@fsv.cuni.cz
http://ies.fsv.cuni.cz

Institut ekonomických studií
Fakulta sociálních věd
Univerzita Karlova v Praze

Opletalova 26
110 00 Praha 1

E-mail : ies@fsv.cuni.cz
http://ies.fsv.cuni.cz

Bibliographic information:
Fičura M., Witzany J. (2023): " Historical Calibration of SVJD Models with Deep Learning " IES Working Papers 36/2023. IES FSV. Charles University.

Historical Calibration of SVJD Models with Deep Learning

Milan Fičura[1]
Jiří Witzany[2]

[1]Faculty of Finance and Accounting, Prague University of Economics and Business, Czech Republic. E-mail: milan.ficura@vse.cz (corresponding author)
[2]Faculty of Finance and Accounting, Prague University of Economics and Business, Czech Republic. E-mail: jiri.witzany@vse.cz

December 2023

Abstract:
We propose how deep neural networks can be used to calibrate the parameters of Stochastic-Volatility Jump-Diffusion (SVJD) models to historical asset return time series. 1-Dimensional Convolutional Neural Networks (1D-CNN) are used for that purpose. The accuracy of the deep learning approach is compared with machine learning methods based on shallow neural networks and hand-crafted features, and with commonly used statistical approaches such as MCMC and approximate MLE. The deep learning approach is found to be accurate and robust, outperforming the other approaches in simulation tests. The main advantage of the deep learning approach is that it is fully generic and can be applied to any SVJD model from which simulations can be drawn. An additional advantage is the speed of the deep learning approach in situations when the parameter estimation needs to be repeated on new data. The trained neural network can be in these situations used to estimate the SVJD model parameters almost instantaneously.

Keywords: Stochastic volatility, price jumps, SVJD, neural networks, deep learning, CNN

Acknowledgement: This paper has been prepared under financial support of a grant GAČR 22-19617S "Modeling the structure and dynamics of energy, commodity and alternative asset prices", which the authors gratefully acknowledge.

1. Introduction

Time-varying volatility is an established feature of financial time series that plays a crucial role in risk management and asset pricing. Unlike GARCH models (Bollerslev, 1986) and realized volatility models (Andersen and Bollerslev, 1998) in which the conditional variance depends deterministically on past returns, stochastic volatility (SV) models (Taylor, 1982) view the variance as an unobservable stochastic process, allowing for great deal of flexibility in its specification (Ghysels et al., 1996).

Log-SV model of Taylor (1982), in which the logarithm of variance follows an AR(1) process can be extended in a variety of ways, including models with correlation between volatility and returns (Harvey and Shephard, 1996), non-Gaussian innovations (Jacquier et al., 2004), price jumps (Andersen et al., 2002), volatility jumps (Eraker et al., 2003), infinite activity jumps (Li et al., 2008), regime switching (So, et al., 1998) or long-memory (Breidt, et al., 1998, Harvey, 1998).

The estimation of SV models is, however, difficult, as their likelihood function involves intractable integrals over the latent volatility process. Wide range of frequentist and Bayesian approaches have been proposed to cope with the estimation problem (Bos, 2012).

Bayesian MCMC sampling (Jacquier, 1994) represents one of the most popular approaches for SV models estimation, with increasingly efficient MCMC samplers developed for most of the basic SV model variants (Kim, Shephard and Chib, 1998, Omori et al., 2007, Nakajima and Omori, 2012, Kastner and Frühwirth-Schnatter, 2014, Hosszejni and Kastner, 2019). Efficient MCMC implementations for some of the SV model variants are available in R packages such as *ASV* (Omori and Hashimoto, 2022) or *stochvol* (Kastner, 2016, Hosszejni and Kastner, 2021, Hosszejni and Kastner, 2023).

A drawback of MCMC is that it is model-specific, requiring major adjustments of the algorithm for any model modification. Among more generic Bayesian approaches is Particle MCMC (Andrieu et al., 2010), and the particle learning methods (Liu and West, 2001, Carvalho, 2010, Fulop and Li, 2013, Chopin et al., 2013). Certain fine-tuning of the particle filters is, however, still needed for good performance.

Frequentist approaches to SV models estimation include moment-based methods, such as GMM (Andersen and Sorensen, 1996) and EMM (Andersen et al., 1999), quasi maximum likelihood (QML) methods (Harvey et al., 1994), Monte-Carlo methods based on Importance Sampling (Sandman and Koopman, 1996, Liesenfeld and Richard, 2003), and methods that approximate the likelihood function with numerical integration (Friedman and Harris, 1998, Watanabe, 1999).

Efficient MLE methods, based on Laplace approximation and automatic differentiation (Skaug and Yu, 2014), or numerical integration (Bégin and Boudreault, 2021) are implemented in the R packages *stochvolTMB* (Wahl, et al., 2021) and *SVDNF* (Mahjoubi et al., 2023) respectively.

While the current repertoire of SV model estimation tools allows for efficient estimation of the most common SV model variants, none of them offers a fully generic method to estimate any type of SV model without additional modifications and fine-tuning of the estimation algorithm. The main contribution of our study is to propose such a generic approach based on deep neural networks (DNN). Our deep learning approach can estimate parameters of any SV model as long as simulations from it can be drawn. As shown in the simulation study, it is also highly accurate and robust, suffering from significantly lower miss-convergence rates than other commonly used methods such as MCMC or MLE. An additional advantage of the DNN approach is its speed in situations where the same

model needs to be re-estimated multiple times as the re-estimation on new data is nearly instantaneous.

The proposed approach is inspired by neural network calibration of SV models for option pricing (Hernandez, 2017). Analogically to the SV model estimation on historical return time series, which is the task of our current study, the calibration of SV model parameters (Heston, 1993, Bates, 1996) to the observed option prices is highly computationally demanding. To simplify the calibration, it was proposed to utilize a neural network (NN) to approximate either the option pricing model (Liu et al., 2019, Horvath et al., 2021, Büchel, et al., 2022), the entire model calibration procedure (Hernandez, 2017, Kim et al., 2023), or the relationship between plain-vanilla and exotic options (Cao et al., 2022), resulting in significant speed gains as well as higher robustness (Cao et al., 2022).

The direct approach (Hernandez, 2017) starts by simulating a large dataset of SV model parameter vectors and calculating selected option prices for each of them. Neural network (NN) is then trained to predict the model parameters based on the set of calculated option prices. Once trained, the NN can be used for near-instantaneous calibration of the SV models to option prices observed on the market.

As shown in Witzany and Fičura (2023a), the NN based calibration of SV models can be successfully applied also on illiquid option markets, with few or even zero observed option prices, by enhancing the NN predictor set with the generalized moments computed from historical asset returns time series. Our paper can be viewed as an extension of this approach, focusing on the case where the SV model needs to be calibrated to the historical time-series of returns only. This can be the case either on markets where no option prices are available or in situations where the parameters of the SV model under the physical probability setting are needed (rather than the risk-neutral ones), such as in risk-management and asset pricing applications.

One of the main complications in the Witzany and Fičura (2023ab) approach is the need to convert the historical time-series of returns to a set of generalized moments sufficient for the SV model estimation, which can be model-specific. To avoid this step, we replace the shallow neural network with a deep neural network, which can be applied to the time series of returns directly, learning the transformations of the return time series needed for the SV model estimation on its own.

We use simulation tests to compare our deep learning approach with the neural network method from Witzany and Fičura (2023a) and with standard statistical approaches based on MCMC implemented in the R package *stochvol* (Kastner, 2016, Hosszejni and Kastner, 2021, Hosszejni and Kastner, 2023) and MLE, implemented in the R package *stochvolTMB* (Skaug and Yu, 2014, Wahl, et al., 2021). The tests confirm that the proposed deep learning approach is highly accurate and robust.

The rest of this paper is organized as follows. Section 2 introduces the baseline SVJD model specification used in our study and the NN/DNN model calibration approaches. Section 3 presents the simulation study comparing the performance NN/DNN methods with MCMC for our baseline SVJD model and with MCMC and MLE for the SV models implemented in the R packages *stochvol* and *stochvolTMB*. Finally, the Conclusion concludes the paper discusses possibilities for future research.

2. Neural Network estimation of SVJD models

For the illustration of our deep learning SV model estimation approach, we use a Gaussian SVJD model with price jumps. This is the baseline model specification on which we illustrate the performance of the NN based calibration method. Robustness of the approach is further assessed for alternative SV model variants from the R packages *stochvol* and *stochvolTMB* in sections 3.3 and 3.4.

2.1. SVJD model specification

Assume that the logarithmic returns $r_t = p_t - p_{t-1}$ follow a discrete-time process:

$$r_t = \mu + \sigma_t \varepsilon_{r,t} + J_t Z_t \quad (1)$$

Where p_t denotes the logarithm of the asset price at time t, r_t is the logarithmic return, μ is the unconditional mean, σ_t is the time-varying volatility, $\varepsilon_{r,t} \sim N(0,1)$ is *i.i.d.* Gaussian white noise, $J_t \sim N(\mu_J, \sigma_J)$ is *i.i.d.* Gaussian variable determining the jump sizes, and $Z_t \sim Bern(\lambda)$is a Bernoulli distributed variable determining the jump occurrences that arrive with intensity λ.

The log-variance $h_t = \log(\sigma_t^2)$ is assumed to follow the AR(1) process (Taylor, 1982):

$$h_t = \alpha + \beta h_{t-1} + \gamma \varepsilon_{h,t} \quad (2)$$

Where $h_{LT} = \alpha/(1-\beta)$ represents the long-term log-variance, β is an AR(1) autoregressive parameter, γ is the volatility of the log-variance and $\varepsilon_{h,t} \sim N(0,1)$ is *i.i.d.* Gaussian white noise assumed to be uncorrelated with $\varepsilon_{r,t}$ (this will be relaxed for the SV models in sections 3.3 and 3.4).

The goal of the estimation procedure is to estimate the vector of parameters $\boldsymbol{\theta} = \{\mu, \alpha, \beta, \gamma, \mu_J, \sigma_J, \lambda\}$ based on the observed time series of asset returns $\boldsymbol{r} = \{r_1, \dots, r_T\}$.

2.2. MCMC for SVJD model estimation

As a benchmark method to estimate the SVJD model in the previous section we use the Markov-Chain Monte-Carlo (MCMC) method (Jacquier, 1994). MCMC allows us to sample from the posterior density $p(\boldsymbol{\theta}^*|\boldsymbol{r})$, where $\boldsymbol{\theta}^*$ denotes a vector of all model parameters and latent states, $\boldsymbol{\theta}^* = \{\boldsymbol{\theta}, \boldsymbol{V}, \boldsymbol{J}, \boldsymbol{Z}\}$, with $\boldsymbol{V} = \ln(\boldsymbol{h})$ denotes the vector of latent stochastic variances, $\boldsymbol{J}$ the vector of latent jumps sizes and $\boldsymbol{Z}$ the vector of latent jump occurrences. Gibbs sampler is used to draw samples from the posterior density $p(\boldsymbol{\theta}^*|\boldsymbol{r})$ by using information about the conditional densities $p(\theta_i^*|\boldsymbol{\theta}_{(-i)}^*, \boldsymbol{r})$. As the conditional density $p(V_i|\boldsymbol{V}_{(-i)}, \boldsymbol{\theta}_{(-v)}^*, \boldsymbol{r})$ is intractable, we use the Accept-Reject Gibbs Sampler proposed in Kim, Shephard and Chib (1998) to sample V_i. The design of the MCMC algorithm is described in Appendix A.

2.3. NN approach to SVJD model estimation

Neural Network (NN) based calibration approach proposed in Witzany and Fičura (2023a) uses a simulated dataset of parameter and return vectors $\{\boldsymbol{\theta}^{(i)}, \boldsymbol{r}^{(i)}\}$ for $i = 1, \dots, N$ in order to train a shallow neural network (Multi-Layer Perceptron, MLP) to estimate the parameter vector $\boldsymbol{\theta}^{(i)}$ based on a set of generalized moments $\boldsymbol{M}^{(i)}$ computed from each simulated return vector $\boldsymbol{r}^{(i)}$. The goal of the neural network is to approximate the function $\boldsymbol{\theta} = f(\boldsymbol{M})$ describing the relationship between the parameter vector $\boldsymbol{\theta}$ and the generalized moments $\boldsymbol{M}$, making the procedure conceptually similar to other moment-based SV model estimation methods (Andersen and Sorensen, 1996, Andersen et al., 1999), but with the calibration procedure replaced with a neural network. The main practical limitation of the approach is the need to expertly specify the set of generalized moments $\boldsymbol{M}$ that contain sufficient information for the estimation of $\boldsymbol{\theta}$**,** which may differ depending on the specification of the SV model. For our study, we selected the generalized moments $\boldsymbol{M}$ used for the estimation of the SVJD model with the NN approach expertly and they are described in Appendix B.

2.4. DNN approach to SVJD model estimation

The main advantage of deep neural networks (DNN) is their ability to learn the set of predictive features from raw data, avoiding the need for expert specification and fine-tuning of the relevant features (LeCun et al., 2015). In our DNN approach to SVJD model calibration we simulate a dataset of

parameter and return vectors $\{\boldsymbol{\theta}^{(i)}, \boldsymbol{r}^{(i)}\}$ for $i = 1, \dots, N$ and then train a Deep Neural Network (DNN) to estimate the parameter vector $\boldsymbol{\theta}^{(i)}$ based on the entire vector of simulated returns $\boldsymbol{r}^{(i)}$. The neural network thus approximates the function $\boldsymbol{\theta} = f(\boldsymbol{r})$ describing the relationship between the parameter vector $\boldsymbol{\theta}$ and the return vector $\boldsymbol{r}$. The principle of the method is thus similar as in the NN approach but with the set of relevant moments learned implicitly by the DNN from the return time series $\boldsymbol{r}$.

While many types of DNN may potentially be used to estimate the function $\boldsymbol{\theta} = f(\boldsymbol{r})$., we focus on the 1-Dimensional Convolutional Neural Networks (1D-CNN) in our study. The 1D-CNN processes the return vector $\boldsymbol{r}$ with a set of 1D convolutional filters followed by (average) pooling layers and global (average) pooling after the last convolutional layer. These convolutional and pooling operations convert the time series $\boldsymbol{r}$ into a set of features $\boldsymbol{M} = g(\boldsymbol{r})$ which are subsequently fed into a set of fully connected layers, in an analogical way as the generalized moments in the NN based approach, but with the relevant moments $\boldsymbol{M} = g(\boldsymbol{r})$ learned implicitly by the 1D-CNN. The need to manually finetune the set of generalized moments $\boldsymbol{M}$ is thus avoided. The architecture of the 1D-CNN used in our study and the settings of the training algorithm are discussed in Appendix C.

3. Simulation tests

3.1. SVJD model estimation – Simulation test design

The goal of this section is to compare the accuracy of the NN/DNN based SVJD model calibration with the MCMC approach used as benchmark. To train the NN/DNN models, we draw a random dataset of $N = 50000$ parameter combinations $\boldsymbol{\theta}^{(i)}$ for $i = 1, \dots, N$ and for each of them simulate a return vector $\boldsymbol{r}^{(i)}$ of length $T = 2000$ by using the SVJD model described in section 2.1. The identical procedure but with dataset size $N_{Out} = 500$ is applied to construct the testing sample. Accuracy of the NN/DNN, trained on the development sample is then compared with the accuracy of the MCMC method on the testing sample. The MCMC is applied only to the testing sample.

To draw the parameter vectors $\boldsymbol{\theta}^{(i)} = \{\mu, v_{LT}, \beta, \gamma, \mu_J, \sigma_J, \lambda\}$ we use the uniform distributions:

- $\mu \sim U(-0.1, 0.1)/250$ (mean daily return) (mju)
- $v_{LT} \sim U(0.005, 0.015)\text{^}2$ (long-term variance of daily returns) (varLT)
- $\beta \sim U(0.79, 0.99)$ (log-variance AR(1) parameter) (beta)
- $\gamma \sim U(0.05, 0.50)$ (volatility of the log-variance) (gamma)
- $\mu_J \sim U(-0.05, 0.05)$ (mean jump size) (mjuJ)
- $\sigma_J \sim U(0.01, 0.10)$ (jump volatility) (sigmaJ)
- $\lambda \sim U(0.005, 0.05)$ (daily jump probability) (lambda)

Where we use the transformation $\alpha = \ln(v_{LT})(1 - \beta)$ to get the parameter α from equation (2). The parameter bounds were set to values that may be realistically observed for financial time series.

The settings of the three estimation methods (MLP, 1D-CNN and MCMC) are as follows:

MCMC: MCMC algorithm described in Appendix A is used with 20 000 MCMC iterations and 10 000 iteration burn-out period. The point estimates are computed as posterior means from the remaining 10 000 MCMC iterations following the burn-out sample.

MLP: Eight variants of the Multi-Layer Perceptron neural network are used, alternatively with 1-4 hidden layers and 20 or 30 neurons in each layer. The MLP networks are trained with the Levenberg-

Marquardt algorithm in Matlab with the default settings. The predictor set of generalized moments $\boldsymbol{M}^{(i)}$ computed for each simulated vector $\boldsymbol{r}^{(i)}$ is described in Appendix B.

1D-CNN: Four 1D Convolutional Neural Network variants are tested, alternatively with 3 or 4 convolutional layers followed by 2 or 3 fully connected layers, each one alternatively with 20 or 30 filters/neurons. Filter width and the pooling width is set to 5 and stride equal to 5 is applied in all pooling layers except the first one. Average pooling is used and the Leaky ReLu activation function is used in the convolutional and fully connected layers. The networks are trained with the Adam algorithm with batch size of 50, layer normalization and early stopping implemented in Matlab. The precise architecture of the tested networks and settings of the training algorithm are discussed in Appendix C.

3.2. SVJD model estimation – Simulation test results

Performance of the individual methods is assessed with the out-sample R-Squared on a testing dataset of newly drawn $N_{Out} = 500$ parameter combinations and simulated return vectors. The values of the out-sample R-Squared for all methods and SVJD model parameters are shown in **Table *1*** where the green color indicates higher accuracy while the red color indicates lower accuracy.

Overall, the 1D-CNN based estimates achieved the highest accuracy for most of the SVJD model parameters except for v_{LT} for which they were (slightly) outperformed by the MLP method. 1D-CNN, on the other hand, significantly outperformed MLP for β, γ and λ. The results are robust with respect to the chosen MLP or 1D-CNN network architecture. On average the MLP networks with more hidden layers slightly outperformed the ones with less hidden layers. For the 1D-CNN the simplest specification with 3 convolutional layers with 20 filters and 3 fully connected layers with 20 neurons achieved the best results for most of the parameters except for β and γ, for which the more complex 1D-CNN with 4 convolutional layers and 3 fully connected layers worked slightly better.

MLP and 1D-CNN methods significantly outperformed the MCMC for all SVJD model parameters. The surprisingly low values of the R-Squared for MCMC seem to be caused mostly by miss-converging chains. This holds in particular for the log-variance AR(1) parameter β for which the MCMC achieved highly negative R-Squared as for some of the simulated time series the MCMC chain converged to a value significantly below the lower bound of the distribution $\beta \sim U(0.79,0.99)$ from which the simulations of β were drawn. We illustrate this issue on

Figure *1* and **Figure *2*** which compare the actual (observed) values of the parameter β for the 500 simulation runs with the predicted values (estimates) of the MCMC and CNN[3x20,2x20] methods respectively.

Table 1 – SVJD model – Simulation test results (out-sample R2)

The table shows out-sample R-Squared of parameter estimates $\boldsymbol{\theta} = \{\mu, \mu_J, \sigma_J, v_{LT}, \beta, \gamma, \lambda\}$ computed with the three alternative model estimation methods (MLP, CNN and MCMC) on a simulated dataset of $N_{Out} = 500$ parameter combinations and return time-series of length $T = 2000$. For MLP the values in [] brackets denote the number of neurons in each layer, while for CNN they denote the number of convolutional layers x filters followed by the number of fully connected layers x neurons.

Model	mju	mjuJ	sigmaJ	varLT	beta	gamma	lambda
MLP[20]	0.4691	0.8269	0.8540	0.9462	0.5425	0.6948	0.6515
MLP[30]	0.4750	0.8367	0.8522	0.9467	0.5473	0.7042	0.6610
MLP[20,20]	0.4764	0.8302	0.8712	0.9480	0.5442	0.7150	0.6700
MLP[30,30]	0.4846	0.8344	0.8659	0.9470	0.5562	0.7149	0.6630
MLP[20,20,20]	0.4902	0.8406	0.8715	0.9419	0.5527	0.7257	0.6756
MLP[30,30,30]	0.4814	0.8327	0.8684	0.9463	0.5584	0.7199	0.6702
MLP[20,20,20,20]	0.4959	0.8427	0.8806	0.9473	0.5575	0.7297	0.6729
MLP[30,30,30,30]	0.4891	0.8413	0.8760	0.9489	0.5494	0.7358	0.6792
CNN[3x20,2x20]	0.5378	0.8674	0.8839	0.9426	0.6600	0.8588	0.7216
CNN[3x30,2x30]	0.4858	0.8669	0.8775	0.9412	0.6717	0.8469	0.7186
CNN[4x20,3x20]	0.5123	0.8551	0.8839	0.9432	0.6877	0.8675	0.7129
CNN[4x30,3x30]	0.5075	0.8562	0.8756	0.9403	0.6789	0.8659	0.7138
MCMC	-0.0269	-0.9123	0.4593	0.8224	-12.4789	0.6950	0.4886

Figure 1 – SVJD model – Out-sample estimates of *beta* produced by the MCMC method

The figure shows out-sample mean posterior estimates of parameter β computed with the MCMC method. It is apparent that while the range of simulated values corresponds to $\beta \sim U(0.79, 0.99)$, MCMC often miss-converged, resulting in estimates that are significantly below the minimum value of 0.79.

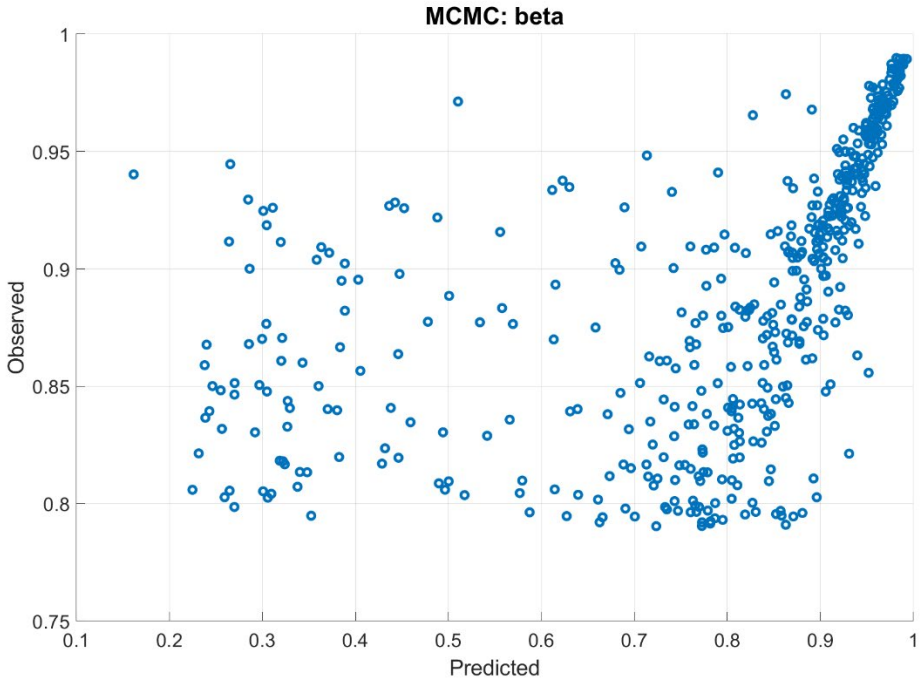
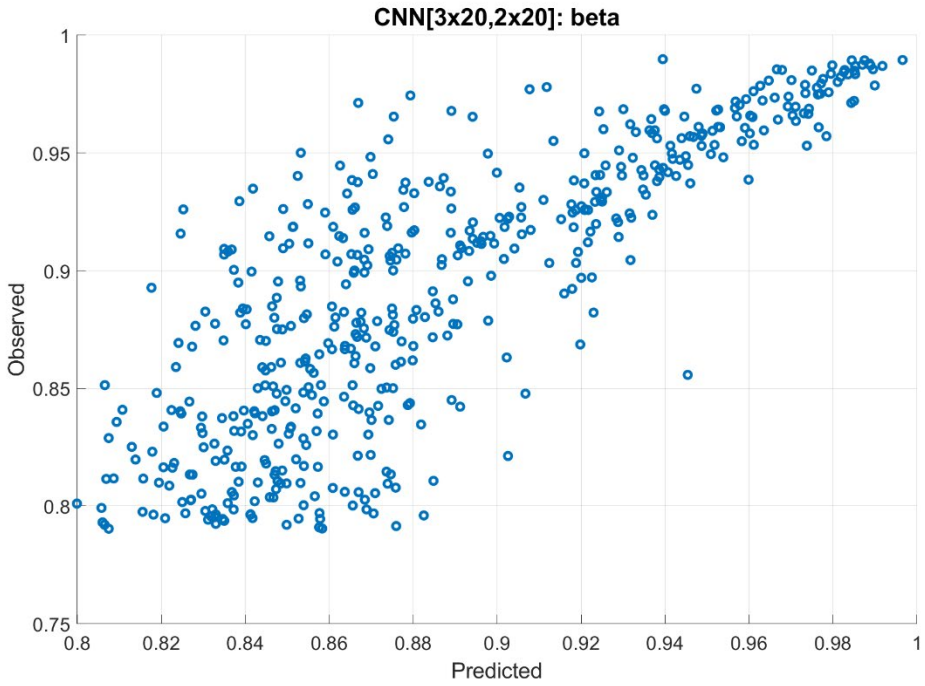


Figure 2 – SVJD model – Out-sample estimates of *beta* produced by the 1D-CNN

The figure shows out-sample estimates of parameter β computed with the CNN[3x20,2x20]. Unlike the MCMC method there are no apparent miss-convergence for any of the simulations with all of the estimates staying in the range of $\beta \sim U(0.79,0.99)$ from which the simulated parameters were drawn.



To isolate the miss-convergence cases, we construct a proxy for miss-convergence by computing the number of parameter estimates that are more than two standard deviations away from their true value for each of the three methods (MLP, CNN and MCMC). The numbers of miss-convergence cases for each parameter and estimation method are shown in **Table 2**. We can see that while the MCMC miss-converged on 125 (e.g. 25%) of the simulated time series for parameter β, the MLP and 1D-CNN methods suffer from almost no miss-convergence cases.

Table 2 – SVJD model – Simulation test results – Number of miss-convergence cases

The table shows the number of simulations in which each of the three estimation methods (MLP, CNN and MCMC) estimated a given parameter from $\boldsymbol{\theta} = \{\mu, \mu_J, \sigma_J, v_{LT}, \beta, \gamma, \lambda\}$ more than two sample standard deviations away from the actual simulated parameter value. The simulation test is based on $N_{Out} = 500$ simulated parameter combinations and return time-series of length $T = 2000$.

Model	mju	mjuJ	sigmaJ	varLT	beta	gamma	lambda
MLP[20]	1	0	0	0	1	1	0
MLP[30]	1	0	0	0	1	1	1
MLP[20,20]	1	1	0	0	1	1	2
MLP[30,30]	2	1	0	0	1	1	1
MLP[20,20,20]	2	0	0	0	1	1	2
MLP[30,30,30]	2	0	0	0	1	1	1
MLP[20,20,20,20]	2	0	0	0	1	1	2
MLP[30,30,30,30]	2	1	0	0	1	1	2
CNN[3x20,2x20]	2	0	0	0	0	0	0
CNN[3x30,2x30]	3	0	0	0	2	0	0
CNN[4x20,3x20]	2	0	0	0	0	0	0
CNN[4x30,3x30]	2	0	0	0	1	0	1
MCMC	24	14	16	2	125	1	8

To verify the performance of the tested SVJD model estimation methods (MLP, CNN and MCMC) more fairly, we recalculate the R-Squared only on the simulations for which convergence was achieved by all estimation methods for all model parameters. In total for 167 simulations at least one method miss-converged (based on the two standard deviation criterion discussed) for at least one of the SVJD model parameters, leaving us with a sample of $N_{Adj} = 333$ simulations for which all methods converged relatively close to the actual parameter values. The values of the R-Squared on these $N_{Adj} = 333$ simulations are shown in **Table *3***.

Table 3 – SVJD model – Simulation test results (out-sample R2) – Corrected for miss-convergence
The table shows out-sample R-Squared of parameter estimates $\boldsymbol{\theta} = \{\mu, \mu_J, \sigma_J, v_{LT}, \beta, \gamma, \lambda\}$, computed with the three alternative model estimation methods (MLP, CNN and MCMC) on a simulated dataset of $N_{Adj} = 333$ parameter combinations and return time-series of length $T = 2000$ for which the estimates of all of the tested models/parameters do not deviate from the true parameters by more than 2 sample standard deviations.

Model	mju	mjuJ	sigmaJ	varLT	beta	gamma	lambda
MLP[20]	0.5183	0.8359	0.8462	0.9309	0.5886	0.6243	0.6847
MLP[30]	0.5195	0.8436	0.8472	0.9306	0.5853	0.6335	0.6975
MLP[20,20]	0.5237	0.8445	0.8636	0.9322	0.5915	0.6472	0.7030
MLP[30,30]	0.5366	0.8440	0.8561	0.9302	0.5950	0.6477	0.6958
MLP[20,20,20]	0.5405	0.8476	0.8647	0.9309	0.5917	0.6555	0.7072
MLP[30,30,30]	0.5341	0.8471	0.8662	0.9304	0.5972	0.6471	0.7021
MLP[20,20,20,20]	0.5449	0.8486	0.8719	0.9318	0.5966	0.6592	0.6966
MLP[30,30,30,30]	0.5359	0.8533	0.8674	0.9325	0.5927	0.6683	0.7087
CNN[3x20,2x20]	0.5979	0.8677	0.8797	0.9222	0.7439	0.8302	0.7525
CNN[3x30,2x30]	0.5335	0.8687	0.8705	0.9281	0.7559	0.8083	0.7505
CNN[4x20,3x20]	0.5519	0.8531	0.8827	0.9272	0.7657	0.8316	0.7494
CNN[4x30,3x30]	0.5496	0.8585	0.8733	0.9215	0.7544	0.8359	0.7539
MCMC	0.2619	0.7732	0.7535	0.9139	0.5533	0.7965	0.6540

We can see that while the MCMC achieves positive R-Squared values once the miss-convergence cases are removed, it is still getting outperformed by the MLP and the 1D-CNN methods (especially for β), with the 1D-CNN achieving the best results for most of the SVJD model parameters (except for v_{LT}).

The results indicate that the 1D-CNN method is able to successfully estimate all of the SVJD model parameters $\boldsymbol{\theta}$ by working with the return vector $\boldsymbol{r}$ alone, avoiding the need to design a model-specific estimation algorithm as in the case of the MCMC, or the need of a fine-tuned set of model-specific generalized moments $\boldsymbol{M}$ as in the case for the MLP approach.

Among the limitations of the performed study is the relatively simple design of the benchmark MCMC algorithm which is mostly based on the algorithms from Jacquier (1994) and Kim, Shephard and Chib (1998) and may not be entirely optimal. Another limitation may be in the use of MCMC mean posterior estimates as a benchmark, which may deviate from the Maximum Likelihood Estimates for asymmetric posterior distributions. Finally, a question remains of whether the proposed 1D-CNN model is indeed fully generic and applicable to any type of SV model including ones with non-Gaussian distributions of innovations and asymmetry (as well as possibly other effects). To partially alleviate these issues and verify the robustness of the 1D-CNN approach we compare its performance with two highly optimized SV model estimation methods, based alternatively on MCMC and MLE, implemented in the popular R packages *stochvol* and *stochvolTMB*. Furthermore, we extend the comparison to all SV model variants that are implemented in these packages. The testing is performed in sections 3.3 and 3.4 respectively.

3.3. SV model estimation – stochvol package

The goal of this section is to verify the robustness of our DNN based SVJD model estimation approach by applying it to the various SV model specifications implemented in the R package *stochvol* (Hosszejni and Kastner 2023) which allows for efficient SV model estimation with modern MCMC algorithms based on the studies of Kastner (2016) and Hosszejni and Kastner (2021).

Package *stochvol* allows for the simulation and estimation of four SV model specifications:

- Log-SV model with Gaussian errors (**SV**)
- Asymmetric log-SV model with Gaussian errors (**ASV**)
- Log-SV model with t-distributed errors (**SV-t**)
- Asymmetric log-SV model with t-distributed errors (**ASV-t**).

The package uses the following SV model notation:

- Return equation: $r_t = e^{h_t/2}\varepsilon_{r,t}$
- Log-variance equation: $h_t = \mu + \phi(h_{t-1} - \mu) + \sigma\varepsilon_{h,t}$

Where r_t denotes the logarithmic return in period t and h_t the logarithmic variance, while $\varepsilon_{r,t}$ and $\varepsilon_{h,t}$ are their error terms whose distribution defines the four possible model specifications:

- SV model → Error terms $\varepsilon_{r,t} \sim N(0,1)$ and $\varepsilon_{h,t} \sim N(0,1)$ are uncorrelated.
- ASV model → Error terms $\varepsilon_{r,t} \sim N(0,1)$ and $\varepsilon_{h,t} \sim N(0,1)$ are correlated with correlation ρ.
- SV-t model → Error terms $\varepsilon_{r,t} \sim T(0,1,\nu)$ and $\varepsilon_{h,t} \sim N(0,1)$ are uncorrelated.
- ASV-t model → Error terms $\varepsilon_{r,t} \sim T(0,1,\nu)$ and $\varepsilon_{h,t} \sim N(0,1)$ are correlated with correlation ρ.

As the asymmetric SV models require a different set of generalized moments than the SVJD model in section 3.1. for the MLP method to provide competitive results (Witzany and Fičura, 2023b), we perform the comparison only between the newly proposed DNN approach and the MCMC method implemented in the *stochvol* package. Additionally, for the sake of sparsity we show the results only for the simplest 1D-CNN architecture from section 3.2. denoted as CNN[3x20,2x20], representing a 1D-CNN with 3 convolutional layers with 20 filters each and 2 fully connected layers with 20 neurons each. The results for other 1D-CNN architectures from section 3.2. are qualitatively similar.

Apart from that, the design of the test is the same as in the previous section. To train the 1D-CNN, we draw a random dataset of $N = 50000$ parameter combinations $\boldsymbol{\theta}^{(i)}$ and for each of them draw a vector of simulated returns $\boldsymbol{r}^{(i)}$ of length $T = 2000$. Similar dataset of size $N_{Out} = 500$ is drawn to get the testing sample on which the accuracy of the 1D-CNN and the MCMC method is compared. The MCMC is run with 20 000 iterations with 10 000 iteration burn-out period and default settings.

Finally, the *stochvol* model parameters are drawn from the following distributions:

- $\mu \sim U(-12,-7)$ (long-term log-variance) (mju)
- $\phi \sim U(0.90,0.995)$ (log-variance AR(1) parameter) (phi)
- $\sigma \sim U(0.05,0.50)$ (volatility of the log-variance) (sigma)
- $\rho \sim U(-0.9,0.9)$ (correlation between log-variance and returns) (rho)
- $\nu \sim U(3,10)$ (degrees of freedom of the t-distribution) (nu)

The results of the simulation test are shown in **Table *4***. The out-sample R-Squared values for all parameters and models are shown in **Panel A**, the miss-convergence rates based on the two standard

deviation criterion are shown in **Panel B**, and the corrected R-Squared values on a subset of simulations for which both methods converged are shown in **Panel C**.

Table 4 – *stochvol* models – Simulation test results (out-sample R2)

The table summarizes the performance of our 1D-CNN estimation approach in comparison with the MCMC algorithms implemented in the R package *stochvol* for four SV model variants: SV, ASV, SV-t and ASV-t. **Panel A** shows the out-sample R-Squared of parameter estimates $\boldsymbol{\theta} = \{\mu, \phi, \sigma, \rho, \nu\}$ computed alternatively with the CNN[3x20,2x20] method or with MCMC method on a simulated dataset of $N_{Out} = 500$ parameter combinations and return time-series of length $T = 2000$. **Panel B** shows the number of simulations for which each of the two methods miss-converged in sense that the model estimate of a given parameter is more than two sample standard deviations aways from the actual value. **Panel C** shows the corrected R-Squared for each model specification once the simulations on which any of the two estimation methods miss-converged (for any of the parameters) are removed from the dataset. This results in $N_{SV} = 432$, $N_{ASV} = 445$, $N_{SV-t} = 349$ and $N_{ASV-t} = 278$ observations for the SV, ASV, SV-t and ASV-t models respectively.

Panel A: Out-sample R-squared on 500 simulated parameter combinations						
Model	**Method**	**mu**	**phi**	**sigma**	**rho**	**nu**
SV	CNN[3x20,2x20]	0.9708	0.7810	0.9447		
	MCMC	0.9658	-14.8476	0.8936		
ASV	CNN[3x20,2x20]	0.9773	0.7855	0.9502	0.9556	
	MCMC	0.4313	-12.4391	0.8334	0.9301	
SV-t	CNN[3x20,2x20]	0.9713	0.7289	0.9200		0.7366
	MCMC	0.9708	-31.9323	0.4499		-2.0128
ASV-t	CNN[3x20,2x20]	0.9670	0.7448	0.9166	0.9347	0.7587
	MCMC	0.8690	-53.5093	-0.8546	0.6482	-4.3263
Panel B: Miss-convergence rates on 500 simulated parameter combinations						
Model	**Method**	**mu**	**phi**	**sigma**	**rho**	**nu**
SV	CNN[3x20,2x20]	0	0	0		
	MCMC	1	67	0		
ASV	CNN[3x20,2x20]	0	1	0	0	
	MCMC	10	45	2	0	
SV-t	CNN[3x20,2x20]	0	0	0		0
	MCMC	0	96	12		91
ASV-t	CNN[3x20,2x20]	0	0	0	0	0
	MCMC	4	131	49	0	154
Panel C: Out-sample R-squared on parameter combinations where both methods converged						
Model	**Method**	**mu**	**phi**	**sigma**	**rho**	**nu**
SV	CNN[3x20,2x20]	0.9671	0.8049	0.9315		
	MCMC	0.9711	0.6360	0.9282		
ASV	CNN[3x20,2x20]	0.9788	0.8167	0.9425	0.9638	
	MCMC	0.9324	0.6278	0.9196	0.9601	
SV-t	CNN[3x20,2x20]	0.9691	0.7370	0.8902		0.7328
	MCMC	0.9695	0.5929	0.8678		0.4914
ASV-t	CNN[3x20,2x20]	0.9589	0.8154	0.8903	0.9470	0.7366
	MCMC	0.9248	0.3270	0.6317	0.6619	0.2125

We can see from **Panel A** that the 1D-CNN outperforms the MCMC algorithms from the package *stochvol* for all tested models and parameters. **Panel B** further shows that while the MCMC miss-converged for up to 25% of the simulated time-series, 1D-CNN miss-converged in only one simulation

(for the ASV model parameter ϕ). Finally, **Panel C** shows that the superior performance of the 1D-CNN persists even once the miss-converging cases are removed from the sample (especially for ϕ and ν).

3.4. SV model estimation – stochvolTMB package

While the DNN approach achieved superior performance against MCMC in the SVJD/SV model estimation tests in sections 3.2. and 3.3., this may potentially be explained by the tendency of MCMC chains to sometimes miss-converge. Additionally, the conversion of MCMC samples into mean posterior estimates may result in a deviation from the Maximum Likelihood Estimates (MLE) if the posterior distributions of the model parameters are asymmetric.

To further verify the robustness of the DNN approach we compare its performance against the approximate Maximum Likelihood (MLE) approach based on Laplace approximation and automatic differentiation (Skaug and Yu, 2014) implemented in the R packages *stochvolTMB* (Wahl, et al., 2021).

Package *stochvoTMB* allows for the simulation and estimation of four SV model specifications:

- Log-SV model with Gaussian errors (**SV**)
- Asymmetric log-SV model with Gaussian errors (**ASV**)
- Log-SV model with skew-Gaussian errors (**SV-sg**)
- Log-SV model with t-distributed errors (**SV-t**).

Compared to the *stochvol*, the package *stochvolTMB* thus allows also for the SV-sg model with skew-Gaussian error distribution but does not allow for an asymmetric version of the SV-t model.

The package uses the following SV model notation:

- Return equation: $r_t = \sigma_r e^{h_t/2} \varepsilon_{r,t}$
- Log-variance equation: $h_t = \phi h_{t-1} + \sigma_h \varepsilon_{h,t}$

Where r_t denotes the logarithmic return in period t and h_t the logarithmic variance, while $\varepsilon_{r,t}$ and $\varepsilon_{h,t}$ are their error terms whose distribution defines the four possible model specifications:

- SV model → Error terms $\varepsilon_{r,t} \sim N(0,1)$ and $\varepsilon_{h,t} \sim N(0,1)$ are uncorrelated.
- ASV model → Error terms $\varepsilon_{r,t} \sim N(0,1)$ and $\varepsilon_{h,t} \sim N(0,1)$ are correlated with correlation ρ.
- SV-sg model → Error terms $\varepsilon_{r,t} \sim SG(0,1,\alpha)$ and $\varepsilon_{h,t} \sim N(0,1)$ are uncorrelated.
- SV-t model → Error terms $\varepsilon_{r,t} \sim T(0,1,\nu)$ and $\varepsilon_{h,t} \sim N(0,1)$ are uncorrelated.

The design of the test is the same as in the previous section. As a representative of the DNN approach we again use the CNN[3x20,2x20] specification. Settings of the MLE are kept at their default values.

The parameter combinations are drawn from the following distributions:

- $\sigma_r \sim U(0.005, 0.045)$ (long-term volatility of returns) (sigma_y)
- $\phi \sim U(0.90, 0.995)$ (log-variance AR(1) parameter) (phi)
- $\sigma_h \sim U(0.05, 0.50)$ (volatility of the log-variance) (sigma_h)
- $\rho \sim U(-0.9, 0.9)$ (correlation between log-variance and returns) (rho)
- $\alpha \sim U(-5, 5)$ (skew par. of the skew-Gaussian distribution) (alpha)
- $\nu \sim U(3, 10)$ (degrees of freedom of the t-distribution) (nu)

The results of the simulation test are shown in **Table *5***. The out-sample R-Squared values for all parameters and models are shown in **Panel A**, the miss-convergence rates based on the two standard

deviation criterion are shown in **Panel B**, and the corrected R-Squared values on a subset of simulations for which both methods converged are shown in **Panel C**.

Table 5 – stochvolTMB models – Simulation test results (out-sample R2)

The table summarizes the performance of our 1D-CNN estimation approach in comparison with the MLE method implemented in the R package *stochvolTMB* for four SV model variants: SV, ASV, SV-sg and SV-t. **Panel A** shows the out-sample R-Squared of parameter estimates $\boldsymbol{\theta} = \{\sigma_r, \phi, \sigma_h, \rho, \alpha, \nu\}$ computed alternatively with the CNN[3x20,2x20] method or with MLE method on a simulated dataset of $N_{Out} = 500$ parameter combinations and return time-series of length $T = 2000$. **Panel B** shows the number of simulations for which each of the two methods miss-converged in sense that the model estimate of a given parameter is more than two sample standard deviations aways from the actual value. **Panel C** shows the corrected R-Squared for each model specification once the simulations on which any of the two estimation methods miss-converged (for any of the parameters) are removed from the dataset. This results in $N_{SV} = 462$, $N_{ASV} = 472$, $N_{SV-sg} = 464$ and $N_{SV-t} = 439$ observations for the SV, ASV, SV-sg and SV-t models respectively.

Panel A: Out-sample R-squared on 500 simulated parameter combinations							
Model	**Model**	**sigma_r**	**phi**	**sigma_h**	**rho**	**alpha**	**nu**
SV	CNN[3x20,2x20]	0.9353	0.7648	0.9234			
	MLE	0.9067	-5.3048	0.8995			
ASV	CNN[3x20,2x20]	0.9379	0.7891	0.9505	0.9634		
	MLE	0.9178	0.1854	0.9354	0.8306		
SV-sg	CNN[3x20,2x20]	0.9370	0.7511	0.9274		0.9689	
	MLE	0.8974	-29.0248	0.7841		0.7815	
SV-t	CNN[3x20,2x20]	0.9386	0.7080	0.9160			0.5175
	MLE	0.8520	-3.5683	0.7643			-8.44E+09
Panel B: Miss-convergence rates on 500 simulated parameter combinations							
Model	**Model**	**sigma_r**	**phi**	**sigma_h**	**rho**	**alpha**	**nu**
SV	CNN[3x20,2x20]	0	0	0			
	MLE	1	37	1			
ASV	CNN[3x20,2x20]	0	0	0	0		
	MLE	2	24	0	9		
SV-sg	CNN[3x20,2x20]	0	0	0		0	
	MLE	3	33	8		1	
SV-t	CNN[3x20,2x20]	0	1	0			0
	MLE	3	34	2			27
Panel C: Out-sample R-squared on parameter combinations where both methods converged							
Model	**Model**	**sigma_r**	**phi**	**sigma_h**	**rho**	**alpha**	**nu**
SV	CNN[3x20,2x20]	0.9312	0.7763	0.9149			
	MLE	0.9086	0.6587	0.9366			
ASV	CNN[3x20,2x20]	0.9340	0.8050	0.9460	0.9685		
	MLE	0.9365	0.7406	0.9505	0.9684		
SV-sg	CNN[3x20,2x20]	0.9421	0.7666	0.9231		0.9691	
	MLE	0.9366	0.6626	0.9422		0.8182	
SV-t	CNN[3x20,2x20]	0.9361	0.7254	0.9084			0.4861
	MLE	0.9011	0.6069	0.9007			0.6012

We can see from **Panel A** that the 1D-CNN outperforms the MLE algorithms from the package *stochvolTMB* for all tested models and parameters, although the results are relatively close, except for ϕ and ν. **Panel B** shows that the miss-convergence rates for MLE are far lower than for the MCMC from section 3.3., but still much higher than for the 1D-CNN which miss-converged in only one simulation for the SV-t model parameter ϕ. The MLE, on the other hand, miss-converged for 24-37

simulations (depending on model specification) for parameter ϕ and for 27 simulations for the SV-t parameter ν. Finally, **Panel C** shows that once all miss-converging cases are removed, the gap between 1D-CNN and MLE performance further narrows, but with 1D-CNN still leading in most of the tests. This is especially apparent for the parameter ϕ. MLE, on the other hand, achieved slightly higher accuracy for parameter σ_r in the ASV model, parameter σ_h in the SV and SV-g models, and parameter ν for the SV-t model. The difference for ν is the most pronounced. We have examined this issue and it seems that the results for ν are relatively volatile with the CNN[3x20,2x20] performance being an outlier. Our empirical tests with other CNN specifications show that the CNN outperform MLE for ν on average.

4. Conclusion

In the study we have shown how deep neural networks (DNN) can be used as a generic method for an accurate and robust estimation of Stochastic-Volatility Jump-Diffusion (SVJD) models on historical data. 1-Dimensional Convolutional Neural Networks (1D-CNN) were used to accomplish this task. The accuracy and robustness of the proposed method was verified on a range of simulation tests that compared the 1D-CNN performance with a similar method based on the Multilayer Perceptron (MLP) neural networks and fine-tuned set of generalized moments, as well as with standard statistical approaches based on MCMC and MLE, including their efficient implementations from the R packages *stochvol* and *stochvolTMB*. The analysis included our workhorse SVJD model as well as other SV model variants including ones with asymmetry and non-Gaussian errors. In all of the tests 1D-CNN achieved the highest performance for almost all of the model parameters, while suffering from significantly lower miss-convergence rates than the MCMC and MLE approaches.

Among the main benefits of the DNN approach is that it is fully generic, allowing for the estimation of any type of SV/SVJD model as long as simulations from the model can be drawn. This is in stark contrast with MCMC which typically requires major modifications and fine-tuning for each SV/SVJD model for which it is applied. It also sets the DNN approach apart from the MLP based methods as while the DNN can be applied directly to the asset return time series, MLP requires the conversion of the return time series into a set of generalized moments (features) which are model-specific and may need to be fine-tuned for each SV/SVJD model variant.

Compared with MLE/MCMC the DNN may also have a speed advantage in situations where the same SV/SVJD model needs to be re-estimated multiple times (e.g. on different time periods or time series), as the re-estimation of the SV/SVJD in such a case is nearly instantaneous.

Among the main weaknesses of the proposed DNN approach is that it produces only point estimates of the model parameters without additional diagnostics or confidence intervals. MCMC, on the other hand, provides their full posterior distributions of the parameters. In situations where additional diagnostics or the full posteriors are needed the DNN thus cannot replace the MCMC.

Finally, while the proposed DNN approach based on 1D-CNN solves the SV/SVJD model estimation problem, it still does not solve the problem of latent states filtering, which is needed to apply the model on out-sample datasets and generate forecasts. The DNN based model estimation procedure would therefore need to be supplemented with a particle filter in order to solve the filtering problem based on the estimated values of the model parameters. As such filters can usually be relatively quickly designed, the DNN still provide a highly generic solution to the SV/SVJD model inference problem. Alternatively, other type of DNN model (such as the LSTM networks) may be used to solve the filtering problem as well, which can be an interesting area of future research.

References

Andersen, T. G., Benzoni, L., Lund, J. (2002). "An Empirical Investigation of Continuous-Time Equity Return Models". *Journal of Finance*, 57(3): 1239-1284

Andersen, T. G., Bollerslev, T. (1998). "Answering the sceptics: yes standard volatility models do provide accurate forecasts". *International Economic Review*, 39 (4): 885–905

Andersen, T. G., Chung, H.J., Sorensen, B.E. (1999). "Efficient method of moments estimation of a stochastic volatility model: A Monte Carlo study". *Journal of Econometrics*, 91(1), 61-87

Andersen, T. G., Sorensen, B.E. (1996). "GMM Estimation of a Stochastic Volatility Model: A Monte Carlo Study". *Journal of Business & Economic Statistics,* 14(3), 328-352

Andrieu C., Doucet, A., Holenstein, R. (2010). "Particle Markov Chain Monte Carlo Methods". *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 72(3), 269–342

Bates, D.S. (1996). "Jumps and Stochastic Volatility: Exchange Rate Processes Implicit in Deutsche Mark Options". *The Review of Financial Studies*, 9(1), 69–107

Bégin, J.F., Boudreault, M. (2021). "Likelihood Evaluation of Jump-Diffusion Models Using Deterministic Nonlinear Filters". *Journal of Computational and Graphical Statistics*, 30(2), 452-466

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

Bos, C.S. (2012). "Relating Stochastic Volatility Estimation Methods." In L Bauwens, C Hafner, S Laurent (eds.), *Handbook of Volatility Models and Their Applications*, John Wiley & Sons, 147–174

Breidt, F. J., Crato, N., de Lima, P. (1998). "The detection and estimation of long memory in stochastic volatility". *Journal of Econometrics*, 83(1-2), 325-348

Büchel, P., Kratochwil, M., Nagl, M., Rösch, D. (2022). "Deep calibration of financial models: turning theory into practice". *Review of Derivatives Research*, 25(2), 109-136

Cao, J., Chen, J., Hull, J., Poulos, Z. (2022). "Deep Learning for Exotic Option Valuation". *The Journal of Financial Data Science*, 4(1), 41-53

Carvalho, C.M., Johannes, M.S., Lopes, H.F., Polson, N.G. (2010). "Particle Learning and Smoothing." *Statistical Science*, 25(1), 88-106

Chopin, N., Jacob, P. E., Papaspiliopoulos, O. (2013). "SMC2: an efficient algorithm for sequential analysis of state space models". *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 75(3), 397-426

Eraker, B., Johannes, M., Polson, N. G. (2003). "The Impact of Jumps in Volatility and Returns". *The Journal of Finance*, 58 (3): 1269-1300

Friedman, M., Harris, L. (1998). "A Maximum Likelihood Approach for Non-Gaussian Stochastic Volatility Models" *Journal of Business & Economic Statistics*, 16(3), 284-291

Fulop, A., Li, J. (2013). "Efficient learning via simulation: A marginalized resample-move approach", *Journal of Econometrics*, 176(2), 146-161

Ghysels, E., Harvey, A. and Renault, E. (1996) "Stochastic Volatility". In: Maddala, G.S. and Rao, C.R., Eds., *Handbook of Statistics (14) Statistical Methods in Finance*, Elsevier, Amsterdam, 119-191

Harvey, A.C., Ruiz, E., Shephard, N. (1994). “Multivariate Stochastic Variance Models”. *The Review of Economic Studies*, 61(2), 247–264

Harvey, A. C., Shephard, N. (1996). “Estimation of an Asymmetric Stochastic Volatility Model for Asset Returns.” *Journal of Business & Economic Statistics*, 14(4), 429–434

Harvey, A. C., (1998). “Long Memory in Stochastic Volatility“, in eds. J. Knight and S. Satchell, *Forecasting Volatility in Financial Markets*, 307-320, Oxford: Butterworth-Heineman.

Hernandez, A. (2017). “Model calibration with neural networks“. *Risk*.

Heston, S.L. (1993). "A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options ". *The Review of Financial Studies*, 6(2), 327–343

Horvath, B., Muguruza, A., Tomas, M. (2021). "Deep learning volatility: a deep neural network perspective on pricing and calibration in (rough) volatility models". *Quantitative Finance*, 21(1), 11–27

Hosszejni, D., Kastner, G. (2019). “Approaches Toward the Bayesian Estimation of the Stochastic Volatility Model with Leverage”. In: Argiento, R., Durante, D., Wade, S. (eds) *Bayesian Statistics and New Generations*. BAYSM 2018. Springer Proceedings in Mathematics & Statistics, 296

Hosszejni, D., Kastner, G. (2021). “Modeling Univariate and Multivariate Stochastic Volatility in R with stochvol and factorstochvol”. *Journal of Statistical Software*, 100(12), 1–34

Hosszejni, D., Kastner, G. (2023). “stochvol: Efficient Bayesian Inference for Stochastic Volatility (SV) Models“, CRAN R package, version 3.2.1

Jacquier, E., Polson, N. G., Rossi, P.E. (1994). “Bayesian Analysis of Stochastic Volatility Models.” *Journal of Business & Economic Statistics*, 20(1), 69–87

Jacquier, E., Polson, N. G., Rossi, P. E. (2004). “Bayesian Analysis of Stochastic Volatility Models with Fat-Tails and Correlated Errors”. *Journal of Econometrics*, 122, 185-212

Kastner, G., Frühwirth-Schnatter, S. (2014). “Ancillarity-sufficiency interweaving strategy (ASIS) for boosting MCMC estimation of stochastic volatility models”. *Computational Statistics & Data Analysis*, 76, 408-423

Kastner, G. (2016). “Dealing with Stochastic Volatility in Time Series Using the R Package stochvol”. *Journal of Statistical Software*, 69(5), 1–30

Kim, S., Shephard, N., Chib, S. (1998). “Stochastic Volatility: Likelihood Inference and Comparison with ARCH Models.” *The Review of Economic Studies*, 65(3), 361–393

Kim, Y.S., Kim, H., Choi, J. (2023). “Deep Calibration with Artificial Neural Network: A Performance Comparison on Option-Pricing Models“. *The Journal of Financial Data Science*, 5(4), 100-118

LeCun, Y., Bengio, Y., Hinton, G. (2015). “Deep learning”. *Nature,* 521, 436–444

Li, H., Wells, M.T., Yu, C.L. (2008). “A Bayesian Analysis of Return Dynamics with Lévy Jumps”. *The Review of Financial Studies*, 21(5), 2345–2378

Liesenfeld, R., Richard, J.F. (2003). “Univariate and multivariate stochastic volatility models: estimation and diagnostics”, *Journal of Empirical Finance*, 10(4), 505-531

Liu, J., West, M. (2001). “Combined parameters and state estimation in simulation-based filtering“. *Sequential Monte Carlo Methods in Practice* (A.Doucet, N. de Freitas and N. Gordon, eds.), Springer, New York, 197-223

Liu, S., Borovykh, A., Grzelak, L.A., Oosterlee, C.W. (2019). “A neural network-based framework for financial model calibration”. *Journal of Mathematics in Industry*. 9(9)

A neural network-based framework for financial model calibration Shuaiqiang Liu, Anastasia Borovykh, Lech A. Grzelak & Cornelis W. Oosterlee Journal of Mathematics in Industry volume 9, Article number: 9 (2019)

Omori, Y., Chib, S., Shephard N., Nakajima, J. (2007). “Stochastic Volatility with Leverage: Fast and Efficient Likelihood Inference”. *Journal of Econometrics*, 140(2), 425–449

Omori, Y., Hashimoto, R. (2022). “ASV: Stochastic Volatility Models with or without Leverage”, CRAN R package, version 1.1.1

Mahjoubi, L.A., Bégin, J.F., Boudreault, M. (2023). “SVDNF: Discrete Nonlinear Filtering for Stochastic Volatility Models”, CRAN R package, version 0.1.8

Nakajima J, Omori Y (2012). “Stochastic Volatility Model with Leverage and Asymmetrically Heavy-Tailed Error Using GH Skew Student’s t Distribution.” *Computational Statistics & Data Analysis*, 56(11), 3690–3704

Sandmann, G., Koopman, S.J. (1998). “Estimation of stochastic volatility models via Monte Carlo maximum likelihood“. *Journal of Econometrics*, 87(2), 271-301

So, M.E.C.P., Lam., K., Li, W.K. (1998). “A Stochastic Volatility Model With Markov Switching”, *Journal of Business & Economics Statistics*, 16(2), 244-253

Skaug, H.J., Yu, J. (2014). “A flexible and automated likelihood based framework for inference in stochastic volatility models”, *Computational Statistics & Data Analysis*, 76, 642-654

Taylor, S.J. (1982) “Financial Returns Modelled by the Product of Two Stochastic Processes—A Study of Daily Sugar Prices 1961-79”. In: Anderson, O.D., Ed., *Time Series Analysis: Theory and Practice 1*, North-Holland, Amsterdam, 203-226

Wahl, J.C. (2021). “stochvolTMB: Likelihood Estimation of Stochastic Volatility Models“, CRAN R package, version 0.2.0

Watanabe, T. (1999). “A non-linear filtering approach to stochastic volatility models with an application to daily stock returns”. *Journal of Applied Econometrics*, 14(2), 101–121

Witzany, J., Fičura, M. (2023a). “Machine Learning Applications for the Valuation of Options on Non-Liquid Option Markets”, SSRN working paper, http://dx.doi.org/10.2139/ssrn.4370426

Witzany, J., Fičura, M. (2023b). “A Comparison of Neural Networks and Bayesian Approaches for the Heston Model Estimation (Forget Statistics – Machine Learning is Sufficient!)”, SSRN working paper, http://dx.doi.org/10.2139/ssrn.4593078

Appendix A – MCMC estimation of the SVJD model

MCMC is used as a benchmark method to estimate the model described in equations (1) and (2).

Our MCMC algorithm proceeds as follows:

0. Initial values of the model parameters were set to: $\mu^{(0)} = 0$, $\alpha^{(0)} = \ln[\mathrm{var}(\boldsymbol{r})] * (1 - 0.9)$, $\beta^{(0)} = 0.9$, $\gamma^{(0)} = 0.3$, $\mu_J^{(0)} = 0$, $\sigma_J^{(0)} = 2 * \mathrm{var}(\boldsymbol{r})$, $\lambda^{(0)} = 0.05$. The initial values of the stochastic variances $\boldsymbol{V}^{(0)}$ were set equal to the exponential moving average of $\boldsymbol{r}^2$ with decay of 0.99, and the initial jump sizes $\boldsymbol{J}^{(0)}$ and jump occurrences $\boldsymbol{Z}^{(0)}$ were set to zero.
1. For $i = 1, \dots, T$ sample jump sizes with $J_i^{(g)} \propto \varphi\left(J; \mu_J^{(g-1)}, \sigma_J^{(g-1)}\right)$ if $Z_i^{(g-1)} = 0$ or with $J_i^{(g)} \propto \varphi\left(r_i; \mu^{(g-1)} + J, \sqrt{V_i^{(g-1)}}\right) \varphi\left(J; \mu_J^{(g-1)}, \sigma_J^{(g-1)}\right)$ if $Z_i^{(g-1)} = 1$.
2. For $i = 1, \dots, T$ sample $Z_i^{(g)} \in \{0,1\}$, with $\Pr[Z = 1] = p_1/(p_0 + p_1)$, where: $p_0 = \varphi\left(r_i; \mu^{(g-1)}, \sqrt{V_i^{(g-1)}}\right)\left(1 - \lambda^{(g-1)}\right)$ and $p_1 = \varphi\left(r_i; \mu^{(g-1)} + J, \sqrt{V_i^{(g-1)}}\right)\lambda^{(g-1)}$.
3. Sample new stochastic log-variances $h_i^{(g)} = \log\left(V_i^{(g)}\right)$ for $i = 1, \dots, T$ with the accept-reject Gibbs sampler (Kim, Shephard and Chib, 1998), by calculating $y_i = r_i - \mu^{(g-1)} - J_i^{(g)} Z_i^{(g)}$ and sampling proposal $h_i^{(g)}$ from $\varphi(h_i; \mu_i, \sigma)$, where: $\mu_i = \phi_i + \frac{\sigma^2}{2}\left[y_i^2 \exp(-\phi_i) - 1\right]$, $\phi_i = \frac{[\alpha(1-\beta) + \beta(\log V_{i+1} + \log V_{i-1})]}{(1+\beta^2)}$ and $\sigma = \frac{\gamma}{\sqrt{1+\beta^2}}$. Proposal $h_i^{(g)}$ is accepted with probability f^*/g^*, where $\ln f^* = -\frac{h_i}{2} - \frac{{y_i}^2}{2}[\exp(-h_i)]$ and $\ln g^* = -\frac{h_i}{2} - \frac{{y_i}^2}{2}[\exp(-\phi_i)\,(1 + \phi_i) - h_i \exp(-\phi_i)]$. If not accepted, a new proposal is drawn (until acceptance).
4. Sample new stochastic volatility autoregression coefficients $\alpha^{(g)}, \beta^{(g)}, \gamma^{(g)}$ from $h_i = \log\left(V_i^{(g)}\right)$ for $i = 1, \dots, T$ using the Bayesian linear regression model: $\widehat{\boldsymbol{\beta}} = (\boldsymbol{X}'\boldsymbol{X})^{-1}\boldsymbol{X}\boldsymbol{y}$, and $\hat{\boldsymbol{e}} = \boldsymbol{y} - \boldsymbol{X}\widehat{\boldsymbol{\beta}}$, where $\boldsymbol{X} = \begin{pmatrix} 1 & \dots & 1 \\ h_1 & \dots & h_{T-1} \end{pmatrix}'$ and $\boldsymbol{y} = (h_2 \dots h_T)'$. Sampling $\left(\gamma^{(g)}\right)^2 \propto IG\left(\frac{n-2}{2}, \frac{\hat{\boldsymbol{e}}'\hat{\boldsymbol{e}}}{2}\right)$ and $\left(\alpha^{(g)}, \beta^{(g)}\right)' \propto \varphi\left[(\alpha, \beta)'; \widehat{\boldsymbol{\beta}}, \left(\gamma^{(g)}\right)^2 (\boldsymbol{X}'\boldsymbol{X})^{-1}\right]$.
5. Sample $\mu^{(g)}$ based on the normally distributed time series $r_i - J_i^{(g)} Z_i^{(g)}$ with variances $V_i^{(g)}$:
$$p\left(\mu^{(g)} | \boldsymbol{r}, \boldsymbol{J}^{(g)}, \boldsymbol{Z}^{(g)}, \boldsymbol{V}^{(g)}\right) \propto \varphi\left(\mu; \sum_{i=1}^{T} \frac{r_i - J_i^{(g)} Z_i^{(g)}}{V_i^{(g)}} \Big/ \sum_{i=1}^{T} \frac{1}{V_i^{(g)}}, \sum_{i=1}^{T} \frac{1}{V_i^{(g)}}\right).$$
6. Sample $\lambda^{(g)}$ from $p\left(\lambda^{(g)} | \boldsymbol{J}^{(g)}\right) \propto \mathrm{Beta}\left(\lambda; n_J^{(g)} + 1, T - n_J^{(g)} + 1\right)$ where $n_J^{(g)}$ denotes the number of realized jumps $n_J^{(g)} = \sum_{i=1}^{T} J_i^{(g)}$.
7. Sample $\mu_J^{(g)}, \sigma_J^{(g)}$ based on the normally distributed series $\boldsymbol{J}^{(g)}$ and uninformative priors $p(\mu) \propto 1$ and $p(\log\sigma^2) \propto 1$, equivalent to $p(\sigma^2) \propto 1/\sigma^2$), from:
$$p\left(\mu_J^{(g)} | \boldsymbol{J}^{(g)}, \sigma_J^{(g-1)}\right) \propto \varphi\left(\mu_J^{(g)}; \frac{\sum_{i=1}^{T} J_i^{(g)}}{T}, \frac{\sigma_J^{(g-1)}}{\sqrt{T}}\right),$$
$$p\left[\left(\sigma_J^{(g)}\right)^2 | \boldsymbol{J}^{(g)}, \mu_J^{(g)}\right] \propto IG\left[\left(\sigma_J^{(g)}\right)^2; \frac{T}{2}, \frac{\sum_{i=1}^{T}\left(J_i^{(g)} - \mu_J^{(g)}\right)^2}{2}\right].$$

Appendix B – Generalized moments used in the MLP approach

To estimate the SVJD model described in equations (1) and (2) with the MLP approach from section 2.2, we had to transform the time-series of returns $\boldsymbol{r}$ into a vector of generalized moments $\boldsymbol{M}$ on which the MLP neural network $\boldsymbol{\theta} = f(\boldsymbol{M})$ could be trained. The generalized moments were chosen expertly based on proxies of volatility, volatility persistence and jumps in financial time series.

The generalized moments used in the empirical study (section 3.1. and 3.2.) are as follows:

1. **Mean return:** $M = \mathrm{mean}(\boldsymbol{r})$
2. **Variance of returns:** $V = \mathrm{var}(\boldsymbol{r})$
3. **Skewness of returns:** $SK = \mathrm{skew}(\boldsymbol{r})$
4. **Kurtosis of returns:** $KS = \mathrm{kurt}(\boldsymbol{r})$
5. **Realized variance:** $RV = \mathrm{mean}(\boldsymbol{r}^2)$
6. **Absolute variation:** $AV = \mathrm{mean}(|\boldsymbol{r}|)$
7. **Bi-power variation:** $BV = \mathrm{mean}\left(\left|\boldsymbol{r}_{1,\dots,T-1}\right| * \left|\boldsymbol{r}_{2,\dots,T}\right|\right)$
8. **Ratio of BV to RV:** $BVRV = BV/RV$
9. **Realized Signed Jumps:** $RSJ = [\mathrm{sum}(\boldsymbol{r}_+^2) - \mathrm{sum}(\boldsymbol{r}_-^2)]/\mathrm{sum}(\boldsymbol{r}^2)$
10. **Mean squared to abs. variation:** $RVAV = \left(RV^{1/2} - AV\right)/RV^{1/2}$
11. **Squared return autocorrelation (lag 1):** $ACF_{sq,1} = \mathrm{acf}(\boldsymbol{r}^2, 1)$
12. **Avg. sq. return autocorrelation (lag 1-5):** $ACF_{sq,1,5} = \mathrm{mean}[\mathrm{acf}(\boldsymbol{r}^2, 1), \dots, \mathrm{acf}(\boldsymbol{r}^2, 5)]$
13. **Avg. sq. return autocorrelation (lag 1-23):** $ACF_{sq,1,23} = \mathrm{mean}[\mathrm{acf}(\boldsymbol{r}^2, 1), \dots, \mathrm{acf}(\boldsymbol{r}^2, 23)]$
14. **Avg. sq. return autocorrelation (lag 1-67):** $ACF_{sq,1,67} = \mathrm{mean}[\mathrm{acf}(\boldsymbol{r}^2, 1), \dots, \mathrm{acf}(\boldsymbol{r}^2, 67)]$
15. **Avg. sq. return autocorrelation (lag 1-252):** $ACF_{sq,1,252} = \mathrm{mean}[\mathrm{acf}(\boldsymbol{r}^2, 1), \dots, \mathrm{acf}(\boldsymbol{r}^2, 23)]$
16. **Squared return autocorrelation (lag 1):** $ACF_{abs,1} = \mathrm{acf}(|\boldsymbol{r}|, 1)$
17. **Avg. sq. return autocorrelation (lag 1-5):** $ACF_{abs,1,5} = \mathrm{mean}[\mathrm{acf}(|\boldsymbol{r}|, 1), \dots, \mathrm{acf}(|\boldsymbol{r}|, 5)]$
18. **Avg. sq. return autocorrelation (lag 1-23):** $ACF_{abs,1,23} = \mathrm{mean}[\mathrm{acf}(|\boldsymbol{r}|, 1), \dots, \mathrm{acf}(|\boldsymbol{r}|, 23)]$
19. **Avg. sq. return autocorrelation (lag 1-67):** $ACF_{abs,1,67} = \mathrm{mean}[\mathrm{acf}(|\boldsymbol{r}|, 1), \dots, \mathrm{acf}(|\boldsymbol{r}|, 67)]$
20. **Avg. sq. return autocorrelation (lag 1-252):** $ACF_{abs,1,252} = \mathrm{mean}[\mathrm{acf}(|\boldsymbol{r}|, 1), \dots, \mathrm{acf}(|\boldsymbol{r}|, 252)]$

Where $\boldsymbol{r}_+^2$ denotes a vector of all positive returns and $\boldsymbol{r}_-^2$ a vector of all negative returns.

Before being used as feature in the MLP neural network, each of the generalized moments was normalized with $\boldsymbol{x}^* = (\boldsymbol{x} - m_{\boldsymbol{x}})/s_{\boldsymbol{x}}$ where $\boldsymbol{x}$ denotes the raw feature, $\boldsymbol{x}^*$ the normalized feature, and the normalization parameters $m_{\boldsymbol{x}} = \mathrm{mean}(\boldsymbol{x})$ and $s_{\boldsymbol{x}} = \sqrt{\mathrm{var}(\boldsymbol{x})}$ estimated on the training dataset of size $N = 50\,000$. The same operations were applied also to each value of the target vector $\boldsymbol{\theta}$.

The normalized moments (features) were used as input into the MLP neural network implemented with the Matlab function **feedforwardnet**. Training was performed with the function **train** using the Levenberg-Marquardt algorithm with default values of its settings.

Appendix C – 1D-CNN architecture

Four 1D Convolutional Neural Network (D-CNN) variants are tested, alternatively with 3 or 4 convolutional layers followed by 2 or 3 fully connected layers, each one alternatively with 20 or 30 filters/neurons. In the result tables we denote these networks as CNN[3x20,2x20], CNN[3x30,2x30], CNN[4x20,3x20] and CNN[4x30,3x30], with the values in [] brackets denoting the number of convolutional layers x filters followed by the number of fully connected layers x neurons.

For all tested networks the filter and pooling width are set to 5 and a stride equal to 5 is applied in all pooling layers except the first one. Average pooling is used and the Leaky ReLu activation function is used in the convolutional and fully connected layers. The choice on the pooling width and the stride is a result of early testing which showed that pooling with stride improves the learning of the SVJD autoregressive parameter, but application of stride in all pooling layers worsens the performance with respect to the volatility of variance parameter and the jump-process parameters. Combined approach where stride is applied to all pooling layers except the first one achieved the most balanced results.

The **CNN[3x20,2x20]** network was initialized as follows:

```
% Set CNN parameters
numFeatures = 1;            % Number of time series used as input
numResponses = size(YN,2);  % Number of parameters of the SVJD model
filterSize = 5;             % Width of the 1D convolutional filters
poolSize = 5;               % Witdh of the 1D pooling operation
numFilters = 20;            % Number of 1D convolutional filters
numNeurons = 20;            % Number of neurons in fully-connected layers

% Construct 3x2 CNN architecture
layers = [ ...
    sequenceInputLayer(numFeatures,MinLength=poolSize*filterSize)
    convolution1dLayer(filterSize,numFilters,Padding="causal")
    leakyReluLayer
    layerNormalizationLayer
    averagePooling1dLayer(poolSize)
    convolution1dLayer(filterSize,numFilters,Padding="causal")
    leakyReluLayer
    layerNormalizationLayer
    averagePooling1dLayer(poolSize,stride=poolSize)
    convolution1dLayer(filterSize,numFilters,Padding="causal")
    leakyReluLayer
    layerNormalizationLayer
    globalAveragePooling1dLayer
    fullyConnectedLayer(numNeurons)
    leakyReluLayer
    layerNormalizationLayer
    fullyConnectedLayer(numNeurons)
    leakyReluLayer
    layerNormalizationLayer
    fullyConnectedLayer(numResponses)
    regressionLayer];
```

The **CNN[3x30,2x30]** architecture is identical to the CNN[3x20,2x20] but with:

```
numFilters = 30;            % Number of 1D convolutional filters
numNeurons = 30;            % Number of neurons in fully-connected layers
```

The architectures of **CNN[4x20,3x20]** and **CNN[4x30,3x30]** differ from the previous ones by adding one more set of convolution and pooling layers before the global average pooling is applied:

```
    convolution1dLayer(filterSize,numFilters,Padding="causal")
    leakyReluLayer
    layerNormalizationLayer
```

Which is added just before the row with `globalAveragePooling1dLayer`.

And one more fully connected layer before the final regression layer:

```
    leakyReluLayer
    layerNormalizationLayer
    fullyConnectedLayer(numResponses)
```

Which is added just before the row with `regressionLayer`.

All networks are trained with the **Adam algorithm** with batch size of 50, layer normalization and early stopping implemented in Matlab. We further set the maximum number of epochs to 100 and the early-stopping patience to 50. The maximum number of epochs was not reached in any of the simulation tests. Relatively high value of the patience was chosen as early tests have shown it tends to improve the out-sample model performance.

The setting of the training algorithm is as follows:

```
% Specify training options
miniBatchSize = 50;     % Mini-batch size
maxEpochs = 100;        % Maximum number of training epochs
vPatience = 50;         % Early-stopping patience parameter
options = trainingOptions("adam", ...
    MiniBatchSize=miniBatchSize, ...
    MaxEpochs=maxEpochs, ...
    SequencePaddingDirection="left", ...
    ValidationData={XNVal_CNN,YNVal}, ...
    ValidationPatience = vPatience, ...
    Plots="training-progress", ...
    Verbose=0);

% Train network
net = trainNetwork(XNTrain_CNN,YNTrain,layers,options);
```

Analogically to the MLP networks, **normalization** of the inputs and outputs of the 1D-CNN was performed before training.

To normalize the inputs, the $N = 50000$ training-sample return vectors $\boldsymbol{r}^{(i)}$ were stacked together into one large return vector $\boldsymbol{R}$ of size $N * T$. The normalization parameters $m_{\boldsymbol{R}} = \mathrm{mean}(\boldsymbol{R})$ and $s_{\boldsymbol{R}} = \sqrt{\mathrm{var}(\boldsymbol{R})}$ were then computed and used to normalize the return vectors $\boldsymbol{r}^{(i)}$ on the training as well as the testing sample with the transformation $\boldsymbol{r}^{*} = (\boldsymbol{r} - m_{\boldsymbol{R}})/s_{\boldsymbol{R}}$.

To normalize the outputs $\boldsymbol{\theta}^{(i)}$ an analogical procedure to the one described in Appending B is applied. For each parameter j, we use the training sample vector of its values $\boldsymbol{\theta}_j$ of size N, to compute the normalization parameters $m_{\boldsymbol{\theta}_j}$ and $s_{\boldsymbol{\theta}_j}$, which are then used to perform the normalization on both the training and the testing sample, $\boldsymbol{\theta}_j^{*} = \left(\boldsymbol{\theta}_j - m_{\boldsymbol{\theta}_j}\right)/s_{\boldsymbol{\theta}_j}$, for all parameters j.

IES Working Paper Series

2023

1. Josef Bajzik, Tomáš Havránek, Zuzana Iršová, Jiří Novák: *Are Estimates of the Impact of Shareholder Activism Published Selectively?*
2. Klára Kantová: *Ex-Prisoners and the Labour Market in the Czech Republic*
3. Theodor Petřík, Martin Plajner: *Concurrent Business and Distribution Strategy Planning Using Bayesian Networks*
4. Tijmen Tuinsma, Kristof De Witte, Petr Janský, Miroslav Palanský, Vitezslav Titld: *Effects of Corporate Transparency on Tax Avoidance: Evidence from Country-by-Country Reporting*
5. Zuzana Irsova, Pedro R. D. Bom, Tomas Havranek, Heiko Rachinger: *Spurious Precision in Meta-Analysis*
6. Vojtěch Mišák: *Does Heat Cause Homicides? A Meta-Analysis*
7. Fan Yang: *The Impact of Regulatory Change on Hedge Fund Performance*
8. Boris Fisera: *Distributional Effects of Exchange Rate Depreciations: Beggar-Thy-Neighbour or Beggar-Thyself?*
9. Salim Turdaliev: *Powering Up Cleaner Choices: A Study on the Heterogenous Effects of Social Norm-Based Electricity Pricing on Dirty Fuel Purchases*
10. Kseniya Bortnikova: *Beauty and Productivity in Academic Publishing*
11. Vladimír Benáček, Pavol Frič: *Ossified Democracy as an Economic Problem and Policies for Reclaiming its Performance*
12. Petr Janský, Miroslav Palanský, Jiří Skuhrovec: *Public Procurement and Tax Havens*
13. Katarzyna Bilicka, Evgeniya Dubinina, Petr Janský: *Fiscal Consequences of Corporate Tax Avoidance*
14. Evžen Kočenda, Shivendra Rai: *Drivers of Private Equity Activity across Europe: An East-West Comparison*
15. Adam Geršl, Barbara Livorová: *Does Monetary Policy Reinforce the Effects of Macroprudential Policy*
16. Tersoo David Iorngurum: *Method versus cross-country heterogeneity in the exchange rate pass-through*
17. T. D. Stanley, Hristos Doucouliagos, Tomas Havranek: *Meta-Analyses of Partial Correlations Are Biased: Detection and Solutions*
18. Samuel Fiifi Eshun, Evžen Kočenda: *Determinants of Financial Inclusion in Africa and OECD Countries*
19. Matej Opatrny, Tomas Havranek, Zuzana Irsova, Milan Scasny: *Publication Bias and Model Uncertainty in Measuring the Effect of Class Size on Achievement*
20. Soňa Sivá: *Effects of Government Interventions on Bank Performance*
21. Oleg Alekseev, Karel Janda, Mathieu Petit, David Zilberman: *Impact of Raw Material Price Volatility on Returns in Electric Vehicles Supply Chain*

22. Karel Janda, Barbora Scherérova, Jan Sila, David Zilberman: *Graph Theory Approach to Prices Transmission in the Network of Commonly Used Liquid Fuels*
23. Yermone Sargsyan, Salim Turdaliev, Silvester van Koten: *The Heterogeneous Effects of Social Cues on Day Time and Night Time Electricity Usage, and Appliance Purchase: Evidence from a Field Experiment in Armenia*
24. Jan Sila, Evzen Kocenda, Ladislav Kristoufek, Jiri Kukacka: *Good vs. Bad Volatility in Major Cryptocurrencies: The Dichotomy and Drivers of Connectedness*
25. Zuzana Irsova, Hristos Doucouliagos, Tomas Havranek, T. D. Stanley: *Meta-Analysis of Social Science Research: A Practitioner's Guide*
26. Diana Kmetkova, Milan Scasny, Iva Zverinova, Vojtech Maca: *Exploring the Link Between Diet and Sustainability in Europe: A Focus on Meat and Fish Consumption*
27. Fisnik Bajrami: *The Impact of Dollarisation on Economic Growth, Investment, and Trade*
28. Miroslav Svoboda, Michael Fanta, Jan Mošovský: *Effectiveness of Car Scrappage Schemes: Comparative Analysis of European Countries*
29. Nicolas Fanta, Roman Horvath: *Artificial Intelligence and Central Bank Communication: The Case of the ECB„*
30. Karel Janda, Jan Sila, David Zilberman: *Fueling Financial Stability: The Financial Impact of U.S. Renewable Fuel Standard*
31. Anna Pavlovova: *High-Frequency Groceries Prices: Evidence from Czechia*
32. Kumar Chandrakamal Pramod Kumar: *Less-cash or more-cash? Determinants and trends of currency in circulation in a panel of 17 economies*
33. Javier Garcia-Bernardo, Petr Janský*:Profit Shifting of Multinational Corporations Worldwide*
34. T. D. Stanley, Hristos Doucouliagos, Tomas Havranek: *Reducing the Biases of the Conventional Meta-Analysis of Correlations*
35. Daniel Bartušek, Evžen Kočenda: *Unraveling Timing Uncertainty of Event-driven Connectedness among Oil-Based Energy Commodities*
36. Milan Fičura, Jiří Witzany: *Historical Calibration of SVJD Models with Deep Learning*

Univerzita Karlova v Praze, Fakulta sociálních věd
Institut ekonomických studií [UK FSV – IES] Praha 1, Opletalova 26
E-mail : ies@fsv.cuni.cz http://ies.fsv.cuni.cz