UDC 004.85
DOI: 10.20535/SRIT.2308-8893.2022.1.08

GENERATIVE TIME SERIES MODEL BASED ON ENCODER-DECODER ARCHITECTURE

N.I. NEDASHKOVSKAYA, D.V. ANDROSOV

Abstract. Encoder-decoder neural network models have found widespread use in recent years for solving various machine learning problems. In this paper, we investigate the variety of such models, including the sparse, denoising and variational autoencoders. To predict non-stationary time series, a generative model is presented and tested, which is based on a variational autoencoder, GRU recurrent networks, and uses elements of neural ordinary differential equations. Based on the constructed model, the system is implemented in the Python3 environment, the TensorFlow2 framework and the Keras library. The developed system can be used for modeling continuous time-dependent processes. The system minimizes a human factor in the process of time series analysis, and presents a high-level modern interface for fast and convenient construction and training of deep models.

Keywords: prediction, variational autoencoder, GRU recurrent neural network, neural ordinary differential equation, latent space, nonstationary time series.

INTRODUCTION

Classical methods of autoregression with moving average (ARMA) [1, 2] are used to analyze and predict stationary time series. Autoregressive models with integrated moving average (ARIMA) [1, 3], heteroskedastic (ARCH/GARCH) [1, 4, 5] and other [6] are designed to analyze a wider class of nonstationary processes. GARCH models, in particular, help to provide the volatility analysis of financial time series [7]. ARIMA models are based on numerical differentiation technique and an operator of finite differences to make time series stationary. Moving variance is applied in GARCH models to model heteroskedasticity. The choice of degree of autoregression and moving average in ARMA, ARIMA, ARCH and GARCH models, when analyzing the autocorrelation, is often carried out manually.

Recurrent neural networks (RNNs) of the long short-term memory (LSTM) type have also been used in recent years to predict time series [8–11]. Gated recurrent unit (GRU), proposed in 2014 [11], is a simplified version of the LSTM network, probably shows as good results as LSTM [12], and therefore is widely used in recent years. Machine learning and deep learning techniques [8–15] mainly require scaling of the input data and presentation of the series in the form of "values for previous periods – values for the current period" or "features – the resulting value". The main problem of such a representation is the invariance of the fixed values of the series with respect to time. This representation of the series assumes that each value of the series is fixed at the same interval, although in practice this is not always the case.

The paper aims to develop a generative model on basis of the autoencoder for time series prediction, which will be sensitive to different intervals of fixing

values of the series and will be able to find hidden patterns in the data. The goal is also to minimize human interference in the data processing, leaving only the requirement to scale the input data.

AUTOENCODER MODELS

An autoencoder is an artificial neural network that, without a teacher, based on an unmarked data is able to recognize encodings – effective representations of input data [8, 13]. Such encodings often have a much smaller dimension compared to the input data, so autoencoders are also a means to reduce dimensionality.

An important feature of the autoencoder is that it can be a *generative model*, capable of randomly generating new data that is very similar to the input. Goals of the autoencoder are as follows: to reconstruct the input data, as well as to identify features hidden in the input data. The typical autoencoder model consists of two parts (Fig.1): the *coder* and the *decoder* networks. The coder has to recognize and convert the input data into a latent space, that is the internal representation of the input data. The decoder, in turn, is seen as a generating network that converts the internal representation into outputs. Typically, the decoder has the same architecture as the coder, but symmetrically mapped relative to the layer responsible for creating the latent space (Fig.1).

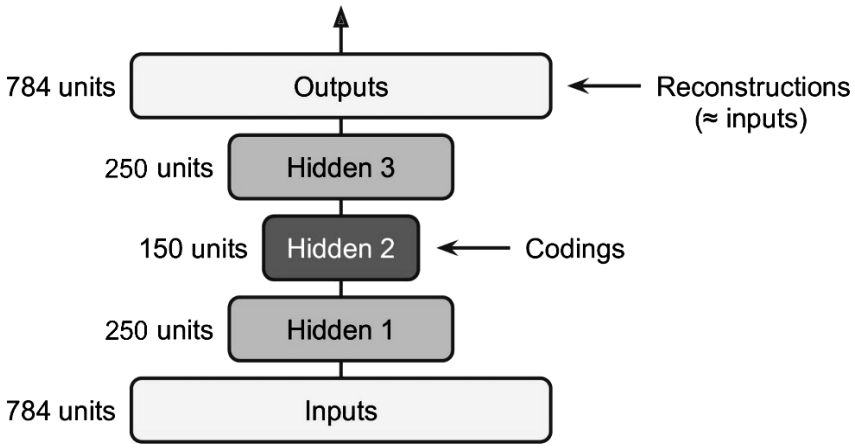

Fig. 1. An example of a deep autoencoder model for the mnist dataset reconstruction, adapted from [8]

To achieve the first goal, namely, to provide the reconstruction of input data, training of the autoencoder is performed by minimizing the loss function, which is called the *reconstruction error:*

$$E(x, g(f(x))), \tag{1}$$

where x is an input vector, E is a function that penalizes $g(f(x))$ for dissimilarity to x, $g(h)$ is the decoder output, $h = f(x)$ is the coder output.

In order to better identify the features hidden in the input data, a regularization is added to the autoencoder model. This allows the model to obtain more properties in addition to the ability to copy input data. The desirable properties of the model are as follows:

– presentation of sparsity of data;

– resistance to noise in the input data and the absence of part of the inputs;

– small values of the derivatives of codings relative to the input data.

The regularized autoencoder is a model with the loss function presented in the form [13]:

$$E(x, g(h)) + \Omega(h, x), \tag{2}$$

where E is the reconstruction loss (1), h is a coder layer, $\Omega(h, x)$ is a coder layer penalty.

The peculiarity of regularized autoencoders is the absence of an obvious Bayesian interpretation. Thus, other known regularized models, for example the ridge regression and other, are an approximation of the Bayesian maximum of the a posteriori probability with the addition of a regularizing penalty, which corresponds to the a priori probability distribution of the model parameters. Regularized autoencoders have a different interpretation, because the $\Omega(h, x)$ penalty depends on the x – an input data and therefore cannot be formally considered as a priori distribution. However, it is still believed that the introduction of the $\Omega(h, x)$ regularizer helps to implicitly prefer certain functions.

Let us consider several models of regularized autoencoders depending on how the penalty $\Omega(h, x)$ in (2) is defined.

Model of sparse autoencoder. One of the key reasons for the high energy efficiency of the human brain is the sparse activation of its neurons: only a small part of the neurons is active in the brain at any given time.

To model the sparseness in an artificial neural network, we consider the probabilistic interpretation of neuronal activation. Let the artificial neuron of the hidden layer be a Bernoulli random variable, and the average value of activation of this neuron corresponds to the probability of obtaining a unit in the Bernoulli test. The probability of activation of each such individual neuron should be low to increase the sparseness of the neurons of the hidden (latent) layer. Let the desired probability of neuron activation be equal to ρ, and let the empirical average value of neuron activation on the basis of train data be equal to $\hat{\rho}$. Sparsity loss [16, 17] is considered as the $\Omega(h, x)$ penalty in expression (2), a measure of dissimilarity between distributions and is based on the Kullback–Leibler divergence between the model distribution and the data distribution:

$$KL(\rho \,\|\, \hat{\rho}) = \rho \ln \frac{\rho}{\hat{\rho}}.$$

Learning criterion for the sparse autoencoder can be considered as follows:

$$(1 - \alpha) E(x, g(f(x))) + \alpha \Omega(h, x),$$

$$\Omega(h, x) = \sum_{i=1}^{N} KL(\rho_i \,\|\, \hat{\rho}_i) = \sum_{i=1}^{N} \rho_i \ln \frac{\rho_i}{\hat{\rho}_i},$$

where $\Omega(h, x)$ is the sparsity loss, N is a number of neurons in a coder layer, $\alpha \in [0,1]$ is a sparsity weight, which is a hyperparameter of the model.

If α is large, the model will pay more attention to the target sparsity, but will not be able to reconstruct the inputs properly. If, on the contrary, the weight is too low, the model will mostly ignore the sparsity and will not find interesting features in the data. Methods of decision support [18, 19] can be used to determine the most acceptable α value based on quantitative and qualitative decision criteria in a particular practical problem.

An important property of the sparse autoencoder is that it can be considered as a generative model with latent variables, which approximates the maximum likelihood. Let us consider a model [13] with an input vector of visible variables *x*, latent variables *h* and a common probability distribution

$$p_{\text{model}}(x,h) = p_{\text{model}}(h)\,p_{\text{model}}(x \mid h).$$

$p_{\text{model}}(h)$ is called the a priori distribution of latent variables and represents the a priori belief of the model that it will "see" the input vector *x*, where *h* is still the output of the coder. This interpretation differs from the traditional use of the term "a priori", which denotes the distribution $p(w)$, which describes the hypotheses about the parameters of the model before reading the training data.

The logarithm of the plausibility of the model can be represented as:

$$\ln p_{\text{model}}(x) = \ln \sum_{h} p_{\text{model}}(x,h).$$

The autoencoder is considered as an approximation of this sum by a point estimate for only one value of *h*, which has a high probability. With this choice of *h*, the following function is maximized:

$$\ln p_{\text{model}}(x,h) = \ln p_{\text{model}}(h) + \ln p_{\text{model}}(x \mid h).$$

The term $\ln p_{\text{model}}(h)$ can cause sparsity. For example, the a priori Laplace distribution

$$p_{\text{model}}(h_i) = \frac{\lambda}{2}\exp(-\lambda \mid h_i \mid)$$

corresponds to the sparsity penalty in terms of the L_1 norm.

Denoising autoencoder. Another way to make the autoencoder show interesting features is to add noise to the inputs and teach it to restore the initial not noisy input [8, 13, 20, 21]. There are two ways:

- a random variable is added to the input vector, normally distributed with a small variance, which determines the noise level;
- part of the input neurons is set to zero. The level of noise is determined by what part it is. This method is more used in image processing problems.

In the denoising autoencoder models, a conditional distribution $C(\hat{x} \mid x)$ of noisy examples under the condition of true examples is introduced. Next, the autoencoder learns the distribution of the reconstruction $p_{\text{reconstr}}(x \mid \hat{x})$, which is estimated on the basis of training pairs $(x,\hat{x})$ as follows [13]:

- select example *x* from the training set;
- select the noisy version $\hat{x}$ with $C(\hat{x} \mid x)$;

- use $(x, \hat{x})$ as a training example to estimate the distribution of reconstruction $p_{\text{reconstr}}(x \mid \hat{x}) = p_{\text{decoder}}(x \mid h)$, where h is the output of the coder, and p_{decoder} is determined by the decoder $g(h)$;
- minimize the following loss function using the mini-batch gradient descent:

$$-\ln p_{\text{reconstr}}(x \mid \hat{x}) = -\ln p_{\text{decoder}}(x \mid h).$$

If the encoder is deterministic, then the autoencoder is often a feedforward neural network, and the same methods can be used to train it as for any feedforward neural network, for example, the mini-batch stochastic gradient descent.

The *variational autoencoder (VAE) model* was proposed in 2014 and is designed to reconstruct the law of distribution of training data for artificial generation of samples from the general distribution [22]. This is a probabilistic model, because its output after training is determined randomly. VAE has a basic architecture common to all autoencoders (Fig. 2): the first part corresponds to the encoder network (it consists of the hidden layers 1 and 2 in the example in Fig. 2), followed by the decoder network (the hidden layers 3 and 4 in Fig. 2). The difference from deterministic encoders is that the VAE encoder for a given input results in the average encoding μ and the standard deviation σ. The coding is then chosen randomly from the Gaussian distribution with mean μ and standard deviation σ. A standard distribution other than Gaussian can also be used. Next, the decoder decodes the received encoding in the usual way.

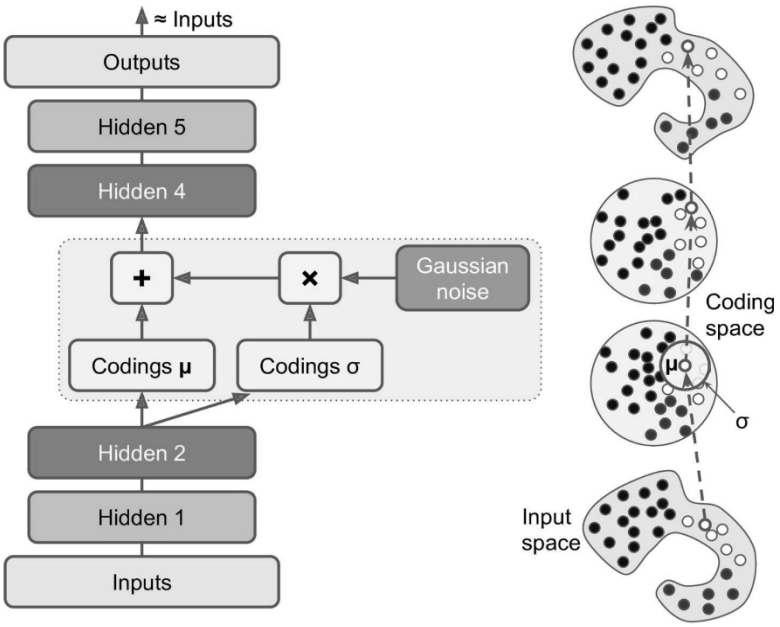

Fig. 2. An example of the variational autoencoder model, VAE [8]

Inputs (on the right in Fig. 2) can have a complex distribution. During training, the coding moves inside the latent space, the coding space, and occupies an approximately spherical region similar to a cloud of Gaussian points. The loss function of the variational autoencoder is the sum of two terms [8]:

$$E(x, g(h)) + L(h, x),$$

where E is the reconstruction error (1) of the input vector x, as before; $L(h,x)$ is the latent loss, which is often the Kullbak–Leibler divergence between the target distribution, such as Gaussian, and the actual coding distribution.

It is easy to generate a new example based on a trained variational autoencoder: you need to choose a random encoding from the Gaussian distribution and decode it.

LODE-GRU-VAE VARIATIONAL AUTOENCODER MODEL FOR TIME SERIES PREDICTION

Fig. 3 shows a simplified architecture of the proposed model. It consists of an encoder – a LODE-GRU network, and a decoder network. Two modifications are proposed: with simulation of timestamp distribution and without it. In both cases, the pair $< x_i, t_i >$ – the data tensor together with the corresponding timestamps is the input for the model (Fig. 3). In the second case we suggest the uniform distribution of these time slices on the observation interval.

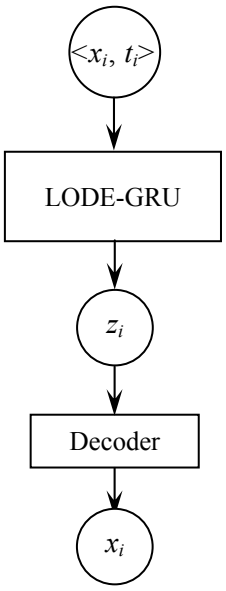

Fig. 3. A simplified LODE-GRU-VAE model

The proposed model is based on the ideas of variational encoder and GRU recurrent neural networks. Features of the GRU unit in comparison with the known LSTM are as follows. Firstly, a single state vector $h(t)$ is used. Secondly, the forgetting gateway and the input gateway are controlled by a single controller – a fully connected layer with a logistical activation function. If the result of this controller is equal to 1, then the forget gateway opens and the input gateway closes. If the result of the controller is equal to 0, then the opposite action occurs. This means that the place is first cleared before saving a certain memory. Thirdly, there is no output gateway in the GRU, so a complete state vector is the result at each time step. The main fully connected layer analyzes the current inputs and some parts of the previous state, which are determined by the additional gateway controller.

The LODE-GRU encoder in Fig. 3 is a modification of recurrent networks of the GRU type. The modification is to use the ordinary differential equations (ODEs) to predict the values of hidden trajectories. As a result, the LODE-GRU network layers are defined as follows:

$$h''_\tau = Sol(f_\theta, h_{\tau-1}, t_{\tau-1}, t_\tau), \tag{3}$$

$$u_\tau = \sigma(W_{xu}x_\tau + W_{hu}h''_\tau + b_u), \tag{4}$$

$$r_\tau = \sigma(W_{xr}x_\tau + W_{hr}h''_\tau + b_r), \tag{5}$$

$$h'_\tau = \tanh(W_{xh'}x_\tau + W_{hh'}(r_\tau \odot h''_\tau) + b_{h'}), \tag{6}$$

$$h_\tau = u_\tau \odot h''_\tau + (1 - u_\tau) \odot h'_\tau. \tag{7}$$

The block described by equation (3) generates a set of points according to the trajectory of the studied process under the conditions (4)–(7) of process transformations, which project the trajectory to another space called latent or hidden. The combination of neural ordinary differential equations and neural RNN networks began to be studied in 2018 to model irregularly observed time series [23], and was further developed in [24–26]. Equation (3) is called the latent ordinary differential equation (Latent ODE). Model (3)–(7) helps to generate new series values at intermediate points between observations, in contrast to standard neural RNN LSTM networks. This is especially useful when the time interval between adjacent observations is large.

Let us modify the described model (3)–(7) to obtain the probability distributions of the hidden trajectories. Let us consider the case when the source vector for the encoder is a multidimensional Gaussian vector with a mathematical expectation equal to μ_{z_0} and the variance equal to σ_{z_0} during the last observation τ. Therefore, the additional layer described by the next formula is added:

$$z_0 = g(h_\tau, \theta) \sim \mathcal{N}(\mu_{z_0}, \sigma_{z_0}). \tag{8}$$

Thus, the LODE-GRU-VAE model is described by formulas (3)–(8) and represents a variation encoder. The coding of the input data x in this model is characterized by a conditional probability distribution $q(z/x)$, where z is a random vector in the latent space. The loss function in this case is the mathematical expectation of losses relative to z:

$$\mathbb{E}_{z\sim q(z/x)} L(z,q) = \mathbb{E}_{z\sim q(z/x)} \ln p(x,z) - D_{KL}(q(z/x) \,||\, p(z)), \tag{9}$$

where $p(x,z)$ is the joint distribution of x and z, $D_{KL}(A \,||\, B)$ is the Kullbak–Leibler divergence, which determines the degree of "dissimilarity" of distributions A and B.

The rationale for the variation encoder is based on the Expectation Maximization (EM) method. For the autoencoder, the input and target are the same vector x. Therefore, the decoder returns the conditional distribution $p(x/z)$ when the code z is an input, and the error function determines the plausibility of the "binding" of the error function to the output of the last layer of the network.

The EM algorithm assumes that we can calculate the distribution $p(x,z/\theta) = p(z/x,\theta)p(x/\theta)$. The problem is to maximize $p(x/\theta)$ with respect to the parameters θ. Let us take the logarithm of both parts of the equality above and express $\ln p(x/\theta)$:

$$\log p(x/\theta) = \ln p(x,z/\theta) - \ln p(z/x,\theta).$$

Next let us take the mathematical expectation with respect to z:

$$\int_Z q(z) \ln p(x/\theta) dz = \int_Z q(z) \ln p(x,z/\theta) dz - \int_Z q(z) \ln p(z/x,\theta) dz.$$

After the transformations we get:

$$\ln p(x/\theta) = \int_Z q(z) \ln\left(\frac{p(x/z,\theta)p(z/\theta)}{q(z)}\right) dz - \int_z q(z) \ln\left(\frac{p(z/x,\theta)}{q(z)}\right) dz.$$

The last term on the right is the Kullbak–Leibler divergence, which is always non-negative quantity. Therefore, the expression $\int\limits_Z q(z)\ln\left(\frac{p(x/z,\theta)p(z/\theta)}{q(z)}\right)dz$ can be considered as the lower estimate of the value of $\ln p(x/\theta)$.

Let us denote $\int\limits_Z q(z)\ln\left(\frac{p(x/z,\theta)p(z/\theta)}{q(z)}\right)dz = ELBO(q,\theta)$. Then

$$ELBO(q,\theta) = \ln p(x/\theta) - D_{KL}(q(z) \,||\, p(z/x)). \quad (10)$$

In general, maximizing (10) by the parameter θ, the approximation $q(z)$ to $p(z/x)$ is performed. Let us show that the loss function (9) is equal to $ELBO(q,\theta)$ (10). Using the formulas of conditional probability, let us rewrite (9) in the form:

$$\mathcal{L}(q) = \mathbb{E}_{z\sim q(z/x)} \ln p(x/z) - D_{KL}(q(z/x) \,||\, p(z)). \quad (11)$$

Since $D_{KL}(q(z/x) \,||\, p(z))$ is close to zero for the same distributions $q(z/x)$ and $p(z)$, and $\ln p(x/z) \le \ln p(x)$, then the upper limit of the loss function will also be $\ln p(x)$. That is, we came to the same result.

The described LODE-GRU-VAE model can be trained by Adam, Rprop or similar methods with respect to the sample $z \sim q(z/x) = q(z; f(x;\theta))$ in order to obtain a gradient with respect to θ and subsequent generation of distribution parameters. With this approximation, the Monte Carlo method can be used to calculate the mathematical expectation for $\ln p(x/z)$ and the Kullbak–Leibler divergence at a fixed θ.

Let us consider how the described LODE-GRU-VAE model can be used to generate a new sequence of series. For example, the initial sequence of n values is first fed to the input of the model, and the prediction of one next value is performed on the basis of the model. Next, the predicted value is joined to the sequence. The prediction for the next value is calculated on basis of the last n values, which are given to the model. This process can generate a new sequence that is similar to the original time series.

STATEMENT AND RESULTS OF THE EXPERIMENT. SELECTION OF MODEL HYPERPARAMETERS

For the time series prediction experiment, let us choose a synthetic nonstationary time series, which is a function of $x = \sin(2\pi\omega t),\ \omega \in \mathbb{R},\ t \in [0,T]$. To bring the data closer to the real ones, the Gaussian noise is added to each value of the series:

$$x = \sin(\pi\omega t) + \epsilon,\ \epsilon \sim \mathcal{N}(0,\sigma),\ \sigma \in \mathbb{R}^+.$$

Not all historical data is always available in real forecasting problems. Therefore, additional values are introduced: d is the sampling step and i is the indicator set of points to be left in the sample. The simulated series can be formally written as follows:

$$\hat{X} = X^{\mathrm{T}} i = \left\| (t_k, \sin(\pi\omega t_k)) \right\|_{k=\overline{1,d}}^{\mathrm{T}} i.$$

The following parameters were selected for the experiment:

- $\omega = 0{,}25$.
- $d = 1000$.
- Number of points to stay is 150.
- The number of layers in the network that simulates the behavior of ODE and their dimension are 1 and 6, respectively.
- Learning rate is adaptive, initially equal to 0,01.
- The method for solving the differential equation is Dormand–Prince.
- The optimization method for learning the neural network is Adamax. Gradients within the ODE-layers are calculated by the method of conjugate equations.
- The dimension of the hidden space is 6.
- The initial value of the variance is 0,1.
- Number of epochs is 200.

The next step is the analysis and processing of the experimental results. Metrics for assessing the quality of the model can be components of the error function. For example, the Kullbak–Leibler divergence can be chosen as a metric. But the limitation of the latter is that it is difficult to interpret. In this work, the mean square error MSE and the coefficient of determination R^2 are used.

Several experiments have been carried out to prove the efficiency of the model (3) – (8) (Fig. 4, 5). In the first experiment, acceptable values of model

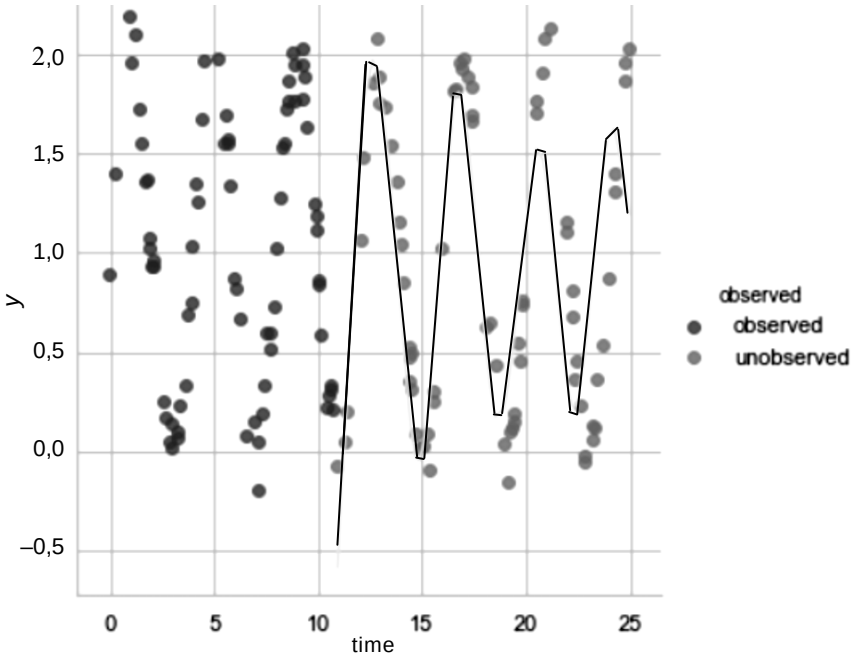

Fig. 4. Real and predicted values of the target variable (experiment 1)

quality metrics were obtained, namely $\mathrm{MSE} = 0{,}11$ and $R^2 = 0{,}78$. In the second experiment, the values of the model quality metrics were almost the same and

also acceptable: $\mathrm{MSE} = 0{,}10$, $R^2 = 0{,}80$. The observed points on which the model was built are marked in Fig. 4, 5 in blue color. The unobserved values of the test set, which were used to assess the quality of the obtained model are marked in orange color. The graph of the predicted values is marked in yellow. Model's predicted values are of admissible MAPE and RMSE rates.

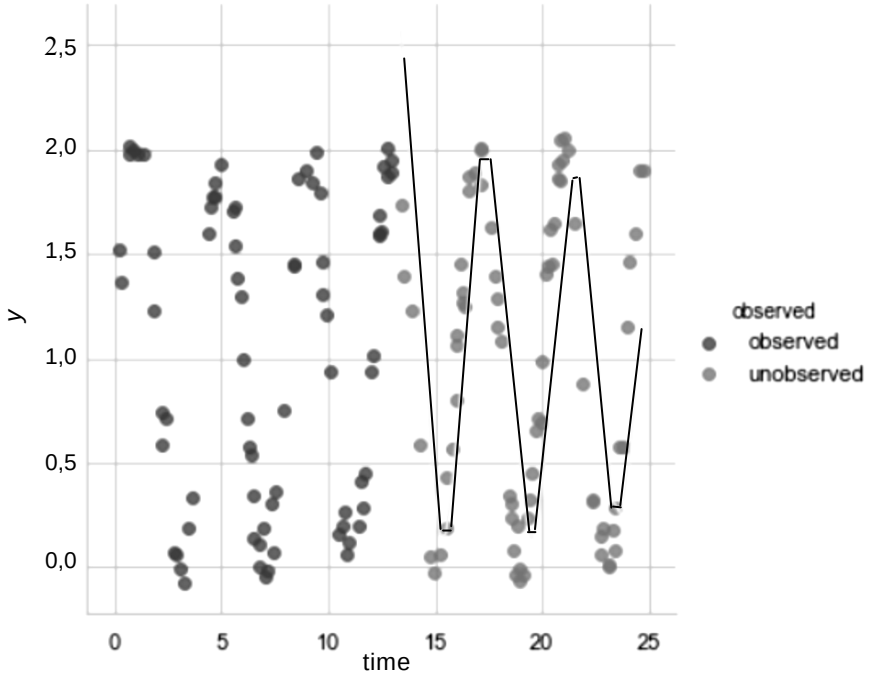

Fig. 5. Real and predicted values of the target variable (experiment 2)

The software in the Python3 environment using the TensorFlow2 framework and the Keras library was developed, which implements the proposed encoder-decoder system. The main arguments in favor of choosing the Python 3 programming language were the speed of writing code and the popularity of this language. Tensorflow2 uses the capabilities of the Nvidia CUDA and has an integrated Keras library, and thus was chosen as the best framework in terms of hardware and performance resources. It is a high-level modern API for fast and easy design and learning of deep models.

CONCLUSIONS

The work is devoted to the research and development of a neural network model based on the encoder-decoder architecture and recurrent blocks for predicting the values of nonstationary time series. Models and methods of machine and deep learning used for sequence processing are studied: LSTM and GRU models of recurrent neural networks, generative models, such as VAE, encoder-decoder models for detecting hidden patterns in data, Adam and Adamax stochastic optimization learning methods.

The LODE-GRU-VAE model was built and tested to reconstruct the dynamics of nonstationary time series. The model allows to generate values at intermediate points between observations, and therefore it is possible to generate new

values of a series where the time interval between two adjacent observations is large. In standard GRU-type RNN networks, the latent state is updated with each observation and remains constant between them. Conversely, within the framework of the LODE-GRU structure, a neural ordinary differential equation learns to model a continuous change in the latent state of a network between two observations.

The encoder-decoder system is implemented based on the proposed model in the Python3 environment using the TensorFlow2 framework and the Keras library. Experiments have been carried out to prove the efficiency of this system in the problems of modeling processes that depend on continuous time.

REFERENCES

1. P.I. Bidyuk, V.D. Romanenko, and O.L. Timoshchuk, *Time series analysis*. Kyiv: Polytechnika, NTUU "KPI", 2013.
2. Terence C. Mills, "Chapter 3 - ARMA Models for Stationary Time Series", *Applied Time Series Analysis. A Practical Guide to Modeling and Forecasting*, pp. 31–56, 2019. Available: https://doi.org/10.1016/B978-0-12-813117-6.00003-X.
3. Terence C. Mills, "Chapter 4 - ARIMA Models for Nonstationary Time Series", *Applied Time Series Analysis. A Practical Guide to Modeling and Forecasting*, pp. 57–69, 2019. Available: https://doi.org/10.1016/B978-0-12-813117-6.00004-1.
4. Amélie Charles and Olivier Darné, "The accuracy of asymmetric GARCH model estimation", *International Economics*, vol. 157, pp. 179–202, May 2019. Available: https://doi.org/10.1016/j.inteco.2018.11.001
5. O.L. Tymoshchuk, V.H. Huskova, and P.I. Bidyuk, "A combined approach to modeling nonstationary heteroscedastic processes", *Radio Electronics, Computer Science, Control*, (2), pp. 80–89, 2019. Available: https://doi.org/10.15588/10.15588/1607-3274-2019-2-9.
6. P. Bidyuk, T. Prosyankina-Zharova, and O. Terentiev, "Modelling nonlinear nonstationary processes in macroeconomy and finances", *Advances in Computer Science for Engineering and Education*, vol. 754, pp. 735–745, 2019. Available: https://doi.org/10.1007/978-3-319-91008-6_72.
7. Anoop S. Kumar and S. Anandarao, "Volatility spillover in crypto-currency markets: Some evidences from GARCH and wavelet analysis", *Physica A: Statistical Mechanics and its Applications*, vol. 524, pp. 448–458,15 June 2019. Available: https://doi.org/10.1016/j.physa.2019.04.154.
8. Aurelien Geron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. Sebastopol, CA: O'Reilly Media Inc., 2017, 760 p.
9. Alex Sherstinsky, "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network", *Physica D: Nonlinear Phenomena*, vol. 404, 132306, March 2020. Available: https://doi.org/10.1016/j.physd.2019.132306.
10. Mikel Canizo, Isaac Triguero, Angel Conde, and Enrique Onieva, "Multi-head CNN–RNN for multi-time series anomaly detection: an industrial case study", *Neurocomputing*, vol. 363, pp. 246–260, 21 October 2019. Available: https://doi.org/10.1016/j.neucom.2019.07.034.
11. Kyunghyun Cho et al., "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation", *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), October 25–29, 2014*, pp. 1724–1734, Available: https://aclanthology.org/D14-1179.pdf.
12. Klaus Greff et al., *LSTM: A Search Space Odyssey*. 2015. Available: arXiv:1503.04069.

13. Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep Learning*. Massachusetts London, England: The MIT Press Cambridge, 2016, 802 p.
14. Henrik Brink, Joseph Richards, and Mark Fetherolf, *Machine Learning*. StPb.: Piter, 2017, 336 p.
15. Jake VanderPlas, *Python Data Science Handbook*. O'Reilly Media, Inc, 2016, 576 p.
16. Z. Deng et al., "Sparse stacked autoencoder network for complex system monitoring with industrial applications", *Chaos, Solitons & Fractals*, vol. 137, August 2020. Available: https://doi.org/10.1016/j.chaos.2020.109838.
17. H. Zhu et al., "Stacked pruning sparse denoising autoencoder based intelligent fault diagnosis of rolling bearings", *Applied Soft Computing*, vol. 88, March 2020. Available: https://doi.org/10.1016/j.asoc.2019.106060.
18. N.I. Nedashkovskaya, "Method for Evaluation of the Uncertainty of the Paired Comparisons Expert Judgements when Calculating the Decision Alternatives Weights", *Journal of Automation and Information Sciences*, vol. 47, no. 10, pp. 69–82, 2015. Available: https://doi.org/10.1615/JAutomatInfScien.v47.i10.70.
19. N.I. Nedashkovskaya, "A system approach to decision support on basis of hierarchical and network models", *System research and information technologies*, no. 1, pp. 7–18, 2018. Available: https://doi.org/10.20535/SRIT.2308-8893.2018.1.01 .
20. J.Yu, "Manifold regularized stacked denoising autoencoders with feature selection", *Neurocomputing*, vol. 358, pp. 235–245, 17 September 2019. Available: https://doi.org/10.1016/j.neucom.2019.05.050.
21. N. Abiri et al., "Establishing strong imputation performance of a denoising autoencoder in a wide range of missing data problems", *Neurocomputing*, vol. 365, pp. 137–146, 6 November 2019. Available: https://doi.org/10.1016/j.neucom.2019.07.065.
22. Diederik P. Kingma and Max Welling, *Auto-Encoding Variational Bayes*, 2014. Available: arXiv:1312.6114v10.
23. Ricky T.Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud, *Neural ordinary differential equations*. NeurIPS, 2018.
24. Yulia Rubanova, Ricky T.Q. Chen, and David K. Duvenaud, *Latent ordinary differential equations for irregularly-sampled time series*. NeurIPS, 2019.
25. Calypso Herrera, Florian Krach, and Josef Teichmann, *Neural Jump Ordinary Differential Equations: Consistent Continuous-Time Prediction and Filtering*, 2020. Available: arXiv:2006.04727.
26. J. Lu et al., "Neural-ODE for pharmacokinetics modeling and its advantage to alternative machine learning models in predicting new dosing regimens", *iScience*, vol. 24, issue 7, 23 July 2021. Available: https://doi.org/10.1016/j.isci.2021.102804.

Received 09.08.2021

INFORMATION ON THE ARTICLE

Nadezhda I. Nedashkovskaya, ORCID: 0000-0002-8277-3095, Institute for Applied System Analysis of the National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Ukraine, e-mail: n.nedashkivska@gmail.com

Dmytro V. Androsov, Institute for Applied System Analysis of the National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Ukraine, e-mail: androsovdmitry80@gmail.com

ГЕНЕРАТИВНА МОДЕЛЬ ДЛЯ ПРОГНОЗУВАННЯ ЧАСОВИХ РЯДІВ НА ОСНОВІ АРХІТЕКТУРИ КОДУВАЛЬНИК-ДЕКОДУВАЛЬНИК / Н.І. Недашківська, Д.В. Андросов

Анотація. Моделі нейронних мереж на основі архітектури кодувальник- декодувальник знайшли широке застосування в останні роки для розв'язання різноманітних задач машинного навчання. Досліджено різновиди таких моделей,

серед яких розріджений, шумопригнічувальний та варіаційний автокодувальники. Для прогнозування нестаціонарного часового ряду подано і протестовано модель, що базується на варіаційному автокодувальнику, блоках рекурентних мереж типу GRU і використовує елементи нейронних звичайних диференціальних рівнянь. На основі побудованої моделі реалізовано систему у середовищі Python3 з використанням фреймворку TensorFlow2 та бібліотеки Keras. Розроблена система може використовуватися для моделювання процесів, що залежать від неперервного часу. Система мінімізує втручання людини у процес аналізу часових рядів, представляє високорівневий сучасний інтерфейс для швидкого і зручного конструювання та навчання глибоких моделей.

Ключові слова: прогнозування, варіаційний автокодувальник, рекурентна нейронна мережа типу GRU, нейронне звичайне диференціальне рівняння, латентний простір, нестаціонарний часовий ряд.

ГЕНЕРАТИВНАЯ МОДЕЛЬ ДЛЯ ПРОГНОЗИРОВАНИЯ ВРЕМЕННЫХ РЯДОВ НА ОСНОВЕ АРХИТЕКТУРЫ КОДИРОВЩИК-ДЕКОДИРОВЩИК / Н.И. Недашковская, Д.В. Андросов

Аннотация. Модели нейронных сетей на основе архитектуры кодировщик-декодировщик нашли широкое распространение в последние годы при решении различных задач машинного обучения. Исследованы разновидности таких моделей, среди которых разреженный, шумоподавляющий и вариационный автокодировщики. Для прогнозирования нестационарного временного ряда представлена и протестирована порождающая модель, которая основана на вариационном автокодировщике, блоках рекуррентных сетей типа GRU и использует элементы нейронных обыкновенных дифференциальных уравнений. На основе построенной модели реализована система в среде Python3 с использованием фреймворка TensorFlow2 и библиотеки Keras. Разработанная система может использоваться для моделирования процессов, зависящих от непрерывного времени. Система минимизирует вмешательство человека в процесс анализа временных рядов, представляет высокоуровневый современный интерфейс для быстрого и удобного конструирования и обучения глубоких моделей.

Ключевые слова: прогнозирование, вариационный автокодировщик, рекуррентная нейронная сеть типа GRU, нейронное обыкновенное дифференциальное уравнение, латентное пространство, нестационарный временной ряд.