Denoising the open-loop step response using an encoder-decoder convolutional neural network

Natalija Đorđević, Nenad Džamić, Aleksa Stojić, Goran Kvaščev

***Abstract*—Denoising signals is used as a preprocessing step for all signal processing. Encoder-decoder neural networks are often proposed as a method of denoising 1D and 2D signals, because of their ability to extract essential features from the signal and then recreate it without noise. In this paper we propose a simple architecture of a convolutional neural network for denoising step responses of systems with different open-loop transfer function. The network is trained on synthetic data with added noise of different distributions, then tested on a portion of synthetic data and real-life step responses.**

***Index Terms*— denoising, encoder-decoder, convolutional neural network, step response**

I. Introduction

Signal denoising is an important preprocessing step in every type of signal processing. Even though many observations can be made on generated pure signals, real-life signals always come with a certain amount of noise. For any signals measured on electrical circuits, noise cause can be the imperfect design or layout of the circuit itself, faulty components, close proximity to other electrical equipment, environmental causes etc. However, even if causes are known, the behavior of noise is unpredictable and rarely fits into a specific probability density function. Having a denoising method that would be applicable to different types of signals and noises would be of great importance for signal processing.

In the paper [1], authors used deep recurrent denoising neural networks to denoise ECG signals and improve signal-to-noise ratio of signal from -8.82dB to 7.71dB. They used a synthetic dataset with added noise. Medical signals such as ECG are of great importance, which shows in the number of papers that deal with their denoising. Once again, a convolutional neural network proves efficient with ECG signals in [2]. Seismic data is one of the representatives of noisy real-life data, and effective use of deep convolutional neural networks for denoising of synthetic seismic data can be seen in [3]. Using a convolutional network for denoising images with Gaussian noise is also proposed in [4]. There is not a single specific architecture for denoising convolutional networks that is superior, and a useful comparison of different kinds is given in [5]. However, one structure seems to be mentioned for multiple purposes, and it is the encoder-decoder structure. The authors in [6] suggest the use of an autoencoder for medical image denoising, which served as an initial inspiration to use encoder-decoder networks for 1D signals. Application of the encoder-decoder structure on denoising micro seismic signals is shown in [7].

In this paper we have trained an encoder-decoder neural network to denoise a step response of an open-loop system. This particular structure seemed promising and in various papers was proven to perform well in denoising for different purposes. Its great advantage is that it does not have strict limitations on the results it can produce. There are however many things that have an effect on its performance, such as the architecture, hyperparameters, the dataset etc. In this paper we attempted to make a structure that is not complicated, but performs well, as well as to generate a dataset that is informative enough for our neural network. The reason that the step response signal was chosen is its importance in observing how industrial processes react to a change in reference value. Without diving into the details of which processes each system represents, we will define their behavior only by their open-loop transfer functions.

II. Dataset synthesis

Collecting data for neural networks can be expensive and time-consuming. In order for our dataset to be as diverse as possible and simultaneously to represent many industrial processes, we used equation (1) as a general form for our open-loop transfer function. [8].

$$G(s) = \frac{e^{-\theta_d s}(1-\alpha s)}{(s+1)^n} \tag{1}$$

Parameter θ_d represents transport delay which is the amount of time that our system needs to react to a change in reference value. The measure $1/\alpha$ represents the position of a zero in the right half-plane. The system in which α is not equal to zero is called the non-minimum phase system and in its step response we can see an initial dip in value before a rise towards the reference value. An example is shown in Fig.1. Finally, the parameter n refers to the order of the system which, in general, dictates the dynamic of the system.

Natalija Đorđević is a teaching associate at the Signals and systems department at the School of Electrical Engineering, University of Belgrade, 73 Bulevar kralja Aleksandra, 11020 Belgrade, Serbia (e-mail: natalija.djordjevic@etf..bg.ac.rs)

Nenad Džamić is with the School of Electrical Engineering, University of Belgrade, 73 Bulevar kralja Aleksandra, 11020 Belgrade, Serbia (e-mail: dn213336m@student.etf.bg.ac.rs)

Aleksa Stojić is a teaching associate at the Signals and systems department at the School of Electrical Engineering, University of Belgrade, 73 Bulevar kralja Aleksandra, 11020 Belgrade, Serbia (e-mail: aleksa.stojic@etf..bg.ac.rs)

Goran Kvaščev is a associate professor at the Signals and systems department at the School of Electrical Engineering, University of Belgrade, 73 Bulevar kralja Aleksandra, 11020 Belgrade, Serbia (e-mail: kvascev@etf..bg.ac.rs)

 ISBN 978-86-7466-930-3

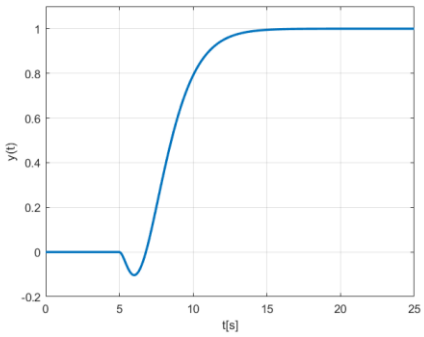

Fig. 1. Step response for a non-minimum phase system with parameters: $\theta_d = 5, n = 3, \alpha = 1$.

Convolutional neural networks are known for needing a large training set to perform well. These parameters can drastically change the output of the system which means that, in order to get a diverse dataset, we should try as many combinations as possible. We generated pure step responses for transfer functions which include parameter values given in Table I. Each signal has a duration of 25s and is sampled with a step size of 0.1s, making it 251 samples long.

TABLE I
TRANSFER FUNCTION PARAMETER VALUES

	Minimum value	Maximum value	Step size
θ_d	0	10	0.2
α	0	2	0.05
n	1	8	1

To every pure signal we generated, we added three types of noise. Given that real-life noise rarely fits into one specific probability density function, feeding different types of noise to our neural network seemed as a good way to have it generalize well after it is trained. The three types of noise added are Gaussian white noise, uniform noise and noise with Rayleigh distribution. All three types of noise have a standard deviation of 0.05. Probability density functions are given in equations (2), (3) and (4), respectively and shown in Fig.2.

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-0.5\left(\frac{x}{\sigma}\right)^2};\ \sigma = 0.05 \tag{2}$$

$$p(x) = \begin{cases} 10/\sqrt{3}, & x \in \{-0.05\sqrt{3}, 0.05\sqrt{3}\} \\ 0, & x \notin \{-0.05\sqrt{3}, 0.05\sqrt{3}\} \end{cases} \tag{3}$$

$$p(x) = \frac{x}{\sigma^2} e^{-x^2/(2\sigma^2)},\ \sigma = 0.05\sqrt{\frac{2}{4-\pi}} \tag{4}$$

The expected value of the Rayleigh distribution given in equation (4) is a positive non-zero value, so to avoid having an offset in our noise we subtracted the expected value.

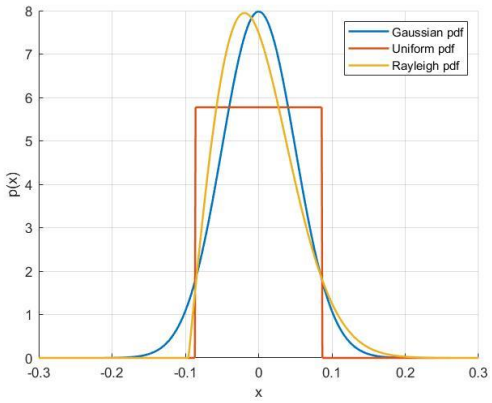

Fig. 2. Comparing three different noise probability density functions.

Final dataset has 50184 samples and 25% percent of it is intended for testing, while the rest is used for training. The test set consists of all step responses for systems of order 4 and 5, while all other orders belong to the training set. This will allow us to test our network on system orders that it has not encountered during training.

III. ARCHITECTURE AND TRAINING OF NEURAL NETWORK

When the term 'denoising' is mentioned in the context of neural networks, the autoencoder neural networks are usually suggested as an appropriate architecture. The idea behind them is to have an encoder part of the network, whose role is to extract features from the given input, followed by the decoder part, which tries to recreate the input from the features. Autoencoder networks are a type of unsupervised learning, because the desired output is the same as the input, therefore the input/output pair is not explicitly given during training. These networks served as an inspiration for this paper and creating the encoder-decoder architecture.

Encoder-decoder neural networks fall into the category of convolutional networks and the term 'convolutional' refers to the method of feature extraction. Let us consider that the input is a 2D matrix, such as a photograph. The dimensions of the photograph tend to be quite large and if we were to feed it to a regular fully-connected neural network, we would have too many trainable parameters. This would imply a huge amount of time to train and a lot of unnecessary computational resources. Also, each pixel itself might not carry as much information about the photograph as their relations do. So instead of treating individual pixels as separate entities, the idea behind convolutional neural networks is to have filters in the form of kernels (matrices with smaller dimensions) whose parameters are to be learned, which in convolution with the input would produce something more informational. Convolution is simply 'sliding' the kernel across the photo and multiplying it with appropriate submatrix of the photo. This way, instead of trying to learn a million neurons, the network has to learn filters which would extract meaningful features. The general advantage of neural networks is that it is not necessary for us to have insight into exactly what these filters are and what features they extract, but only how well the output is decoded using those features.

 ISBN 978-86-7466-930-3

Convolution can also be done with 1D inputs, such as various forms of signals. The principal of convolution stays the same and the kernel is now a vector instead of a matrix. As previously mentioned, the aim of this project is to use a convolutional encoder-decoder network to denoise a step response of an open-loop system.

For this purpose we used *python* and the *tensorflow* library. It allows users to build a sequential neural network model by adding desired layers one-by-one.

For the encoder part of the network, we followed a standard structure of convolutional networks which implies alternatively adding convolutional layers and pooling layers. Convolutional layers learn kernel parameters and produce a convolution of the input with the kernel. It is common practice to have kernels with small dimensions, so in every convolutional layer we used kernels of size 5 (vectors of length 5). Every convolutional layer we added is followed by a *max-pooling* layer, whose role is to cut down the dimensions. It takes subvectors (of the specified size) of the input and produces an output which is the maximum value of the subvector. If the size is properly defined, we get the effect of having less computation needed, without any crucial loss in information.

The decoder part of the network has the inverse structure of the encoder part. Therefore, all convolutional layers are replaced with transposed convolutional layers. They perform deconvolution, which is an inverse operation to convolution. Opposite of *max-pooling* layers are *upsampling* layers. They once again make the dimensions larger by repeating the values of their input a specified amount of times. It is easy to notice that the output of the *upsampling* layers will not be exactly the same as the input of *max-pooling* layers.

The final architecture we chose is the simplest one that performed well. Details of its structure are shown in Table II. and Fig. 3.

TABLE II
ARCHITECTURE OF NEURAL NETWORK

Layer type	Kernel size	Output shape
Convolutional	5	(1, 247, 128)
Max-pooling	2	(1, 124, 128)
Convolutional	5	(1, 120, 64)
Max-pooling	2	(1, 60, 64)
Transposed convolutional	5	(1, 64, 64)
Upsampling	2	(1, 128, 64)
Transposed convolution	5	(1, 132, 128)
Upsampling	2	(1, 264, 128)
Convolutional	5	(1, 264, 1)
Cropping	/	(1, 251, 1)

The activation function for all layers is ReLU, except for the output layer whose activation function always depends on what the network is trying to predict. Since in our case we are doing regression, the adequate activation function is linear. The metric used to access the performance is mean-squared error, also suitable for regression.

When training a neural network it is necessary to make sure not to overfit it to the training data. Overfitting is a term used to describe the behaviour of a network that performs very well on training data, but has problems with data that it has not encountered during training, i.e. it generalizes poorly. One method to prevent overfitting we already implicitly implemented by making a large training dataset. Having a diverse dataset with many samples decreases the chances of overfitting. We also added a kernel constraint with value 3 to every layer, which stops parameters from rising above the given value. Popular method to avoid overfitting is also adding *dropout* layers after convolutional layers. Their role is to remove certain neurons with a specified rate and have them not impact the output. Adding *dropout* layers did not make the network perform better, so they were left out. 20% of samples in the dataset were used for validation. Finally, the network was trained for 7 epochs in mini-batches of size 32.

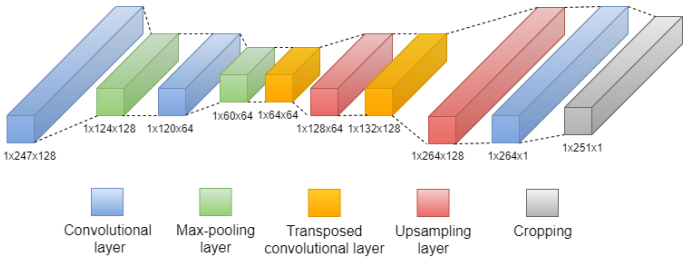

Fig. 3. Neural network architecture

The optimizer used is *Adam* optimizer, which is a type of a stochastic gradient descent algorithm. It is known for keeping track of the first and second momentum to provide quicker and more stable convergence. Its primary hyperparameters are learning rate, β_1 and β_2. We used stochastic grid-search to find optimal hyperparameter values and they are: learning rate = 0.0005, $\beta_1 = 0.9$ and $\beta_2 = 0.99$.

IV. RESULTS

The network was first tested on a generated test set mentioned in chapter II. Average mean-squared error (mse) on the entire test set and on separate noise types are shown in Table III. In Fig. 5 the average values and standard deviations of mse for each of the noises are shown. Gaussian noise has the highest error, however all three have values bellow $9e^{-5}$. One more metric to asses the denoising process is comparing the signal-to-noise ratios (SNR) of signals before and after denoising. The average SNR for all types of noise was 53.5dB in the beginning. Signal-to-noise ratios after denoising are given in Table III.

 ISBN 978-86-7466-930-3

TABLE III
MEAN-SQUARED ERROR FOR DIFFERENT NOISE TYPES

	MSE	SNR (dB)
Gaussian noise	$8.84e^{-5}$	87.8
Uniform noise	$8.14e^{-5}$	88.6
Rayleigh noise	$7.21e^{-5}$	89.9
Entire test set	$8.06e^{-5}$	88.8

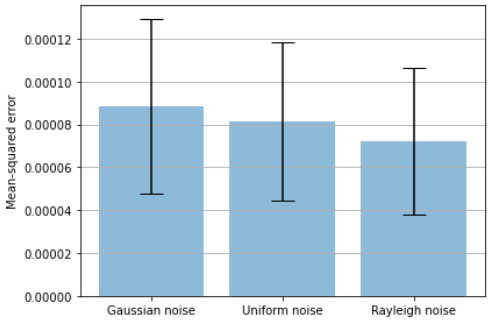

Fig. 4. Means and standard deviations of mean-squared errors for each noise

In Fig 5A. there are three representative step responses: one for each of the three types of noises. Based on observing, the output of the network seems to follow the signal dynamic well. Judging by the mean-squared error and Fig. 5A, Gaussian noise is the most difficult to remove, while Rayleigh noise seems the easiest to remove.

Next step was to test our network in a real-world environment to see how it handles a step response with noise which does not specifically match any distribution. The system we tested on was a handmade dryer that consists of a chamber on a metal surface and it has three platinum temperature sensors and one sensor which measures the airflow. The chamber is 1m long and has a heater of 400W and a ventilator on one side. Since airflow is not steady and turbulence occurs, the signal we captured is very noisy. In Fig 5B. there are three different original signals of airflow, as well as their denoised version. The results are satisfactory considering the amount of noise and the fact that the exact transfer function was not included in the training set.

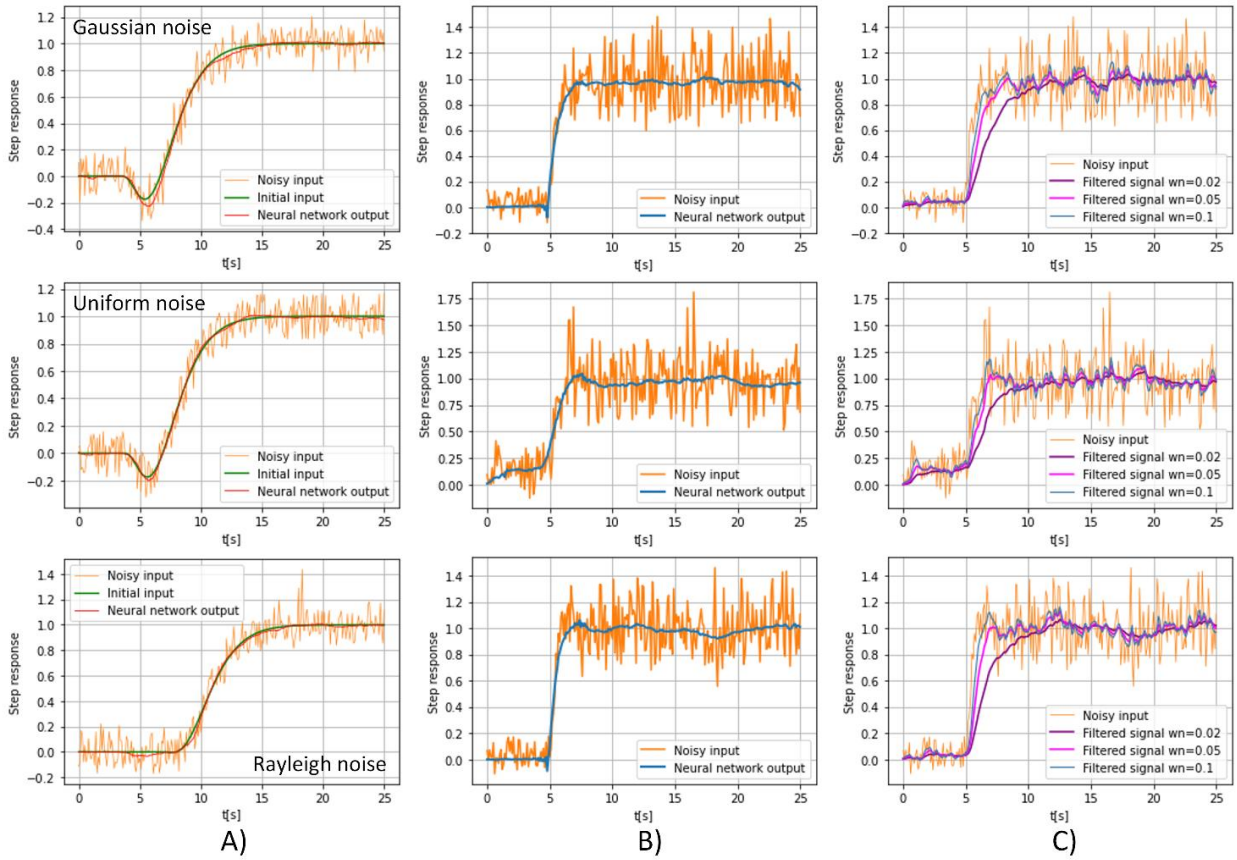

Fig. 5. Step responses before and after denoising. A) synthetic data, B) airflow signal after neural network, C) airflow signal filtered with Butterworth filter

To compare our results with classical filtering methods, we give the same noisy inputs but filtered with a first order Butterworth filter with different normalized frequencies in Fig. 5C. We see that in order to not compromise the dynamic of the signal, we have to settle for less noise removal. The neural network we trained gives better results in terms of removing noise, but still following the dynamic. The comparison we did gives us an insight into why using a neural network for denoising could be a justified option. Conventional filtering methods often have to make sacrifices in terms of degrading

 ISBN 978-86-7466-930-3

some other signal characteristics. Even though the structure of the neural network is not necessarily simple for all purposes, it is very flexible, and if we feed it the right amount of input data to learn from, the results it yields can be satisfactory in many ways.

V. Conclusion

The architecture of neural network used in this paper shows great potential for signal denoising. The proposed general form of transfer function in equation (1) also proved to be enough for the network to be able to follow the step response dynamic well. The next step in improving performance and extending it to other signals can move in a couple of directions. Having a training set with more noise distributions would also help the network generalize even better. Next possible extension could be having step responses with added disturbances at different moments in time in our dataset, as well as creating open-loop transfer functions in a more generalized form.

Acknowledgment

The paper was co-funded by the Ministry of Education, Science, and Technological Development of the Republic of Serbia. This year's contract number is 451-03-68/2022-14/200103.

References

[1] Antczak, Karol. "Deep recurrent neural networks for ECG signal denoising." *arXiv preprint arXiv:1807.11551* (2018).

[2] Arsene, Corneliu TC, Richard Hankins, and Hujun Yin. "Deep learning models for denoising ECG signals." *2019 27th European Signal Processing Conference (EUSIPCO)*. IEEE, 2019.

[3] Zhu, Weiqiang, S. Mostafa Mousavi, and Gregory C. Beroza. "Seismic signal denoising and decomposition using deep neural networks." *IEEE Transactions on Geoscience and Remote Sensing* 57.11 (2019): 9476-9488.

[4] Murali, Vineeth, and P. V. Sudeep. "Image denoising using DnCNN: an exploration study." Advances in Communication Systems and Networks. Springer, Singapore, 2020. 847-859.

[5] Thakur, R. S., Yadav, R. N., & Gupta, L. (2019). State-of-art analysis of image denoising methods using convolutional neural networks. *IET Image Processing*, *13*(13), 2367-2380.

[6] Gondara, Lovedeep. "Medical image denoising using convolutional denoising autoencoders." *2016 IEEE 16th international conference on data mining workshops (ICDMW)*. IEEE, 2016

[7] Zhang, Hang, et al. "Microseismic signal denoising and separation based on fully convolutional encoder–decoder network." *Applied Sciences* 10.18 (2020): 6621.

[8] Goran Kvaščev. "Dalji razvoj i uporedna analiza procedura za eksperimentalno projektovanje i podešavanje industrijskih regulatora" Magister's thesis, 2005.

 ISBN 978-86-7466-930-3