Research and Development of Data Compression Methods Based on Neural Networks

A. Berezkin(✉), D. Kukunin, and R. Kirichek

Bonch-Bruevich Saint-Petersburg State University of Telecommunications, Bolshevikov Ave. 22 build. 1, 193232 St. Petersburg, Russia
aa.berezkin@mail.ru , kirichek@sut.ru

Abstract. This article discusses a neural network-based compression algorithm using error-correcting codes. The use of this algorithm has a number of advantages, on the one hand, the noise-resistant code allows to get rid of potentially high overheads provoked by the use of a neural network, lowering the required value of model accuracy to the value determined by the correctability of the used code. On the other hand, a trained neural network allows data compression without prior transformations.

Keywords: Autoencoder · Data compression · Neural networks

1 Introduction

Nowadays, the emergence of new types of communication networks, particularly, networks using tethered and autonomous UAVs, has led to the development of different data transmission methods [1,7–9]. As expected in the future networks, it is required to consider having an ultra-low latency and high bandwidth for some applications such as augmented/virtual reality or tactile internet [6,13]. The modern stage of telecommunications development imposes stringent requirements for speed, delay and reliability in transmitting and processing information.

One of the approaches to increase the speed of data transmission, and hence to eliminate delays, is the compression of information in communication channels on the fly. This approach requires the development of new information processing architectures that allow data processing in parallel. Neural networks by virtue of their structure can fulfill the set requirements.

Neural networks are actively used in image and other compression tasks [2,10–12], since the generalization ability of this method allows to achieve lossy compression. Theoretically, it is possible to achieve full data recovery [3], but in practice developers face high overhead costs associated with an increase in the occupied space of the network itself, as well as, with an increase in the computational complexity of the algorithm. The proposed codec structure is shown in Fig. 1, where the neural network part of the codec is highlighted by a rectangle.

V. M. Vishnevskiy et al. (Eds.): DCCN 2022, LNCS 13766, pp. 103–116, 2022.
https://doi.org/10.1007/978-3-031-23207-7_9

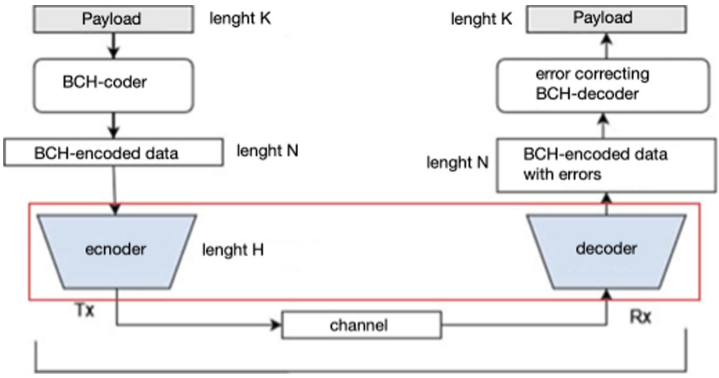

Fig. 1. Neural network codec architecture

This architecture consists of several independent processing units:

– Interference code encoder and decoder;
– A compression algorithm based on a neural network;

The use of this architecture has several advantages, on the one hand, the error-correcting code [3,4] allows to get rid of the potentially high overheads induced by the use of the neural network, lowering the required value of the final model accuracy to the value determined by the corrective capability of the used noise-correcting code. On the other hand, a trained autoencoder neural network allows data compression without prior transformations.

2 Development of a Research Stand

In this paper we investigated the operation of the autoencoder using several error-correcting codes of different multiplicity. The following error-correcting codes were used: (31, 21), (31, 16) and (31, 5).

These codes have different correction ability, as well as different number of information bits, which will allow to better explore the capabilities of the neural network and determine the dependence of the hidden state size not only on the number of guaranteed correctable errors, but also to determine the influence of the code distance on the generalization ability of the autoencoder neural network.

The criterion for comparing different architectures is the fact of lossless data recovery from the hidden state. This metric can be described as the accuracy of the neural network and defined as the fraction of correctly recovered symbols relative to the whole set. For a binary dataset, the accuracy formula is as follows:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN'}, \tag{1}$$

where $accuracy$ is the accuracy of the autoencoder, TP is the number of correctly recovered units, TN is the number of correctly recovered zeros, FP is the number of incorrectly recovered units, FN is the number of incorrectly recovered zeros.

For lossless data recovery, the accuracy should be equal to one. The accuracy shows the quality of the neural network with respect to the whole data set, not paying attention to errors in specific code combinations. In practice, there may be situations in which the neural network will perform well relative to the full data set, but some of the code combinations cannot be recovered in their original form. To control the accuracy of the autoencoder at the level of a particular code combination, an additional metric was introduced, defined as the minimum accuracy of the neural network relative to the code combination.

$$CodeWiseAcc = min(accuracy(X_{i=1..n})), \tag{2}$$

where $CodeWiseAcc$ is minimum accuracy with respect to code combination, $accuracy$ is accuracy metric, $X_{i=1...n}$ is recovered element of original sequence. This metric allows to trace the maximum value of possible errors in a separate code combination and gives a better idea about the restoring capability of autoencoder.

The following formula was used to calculate the lower bound of autoencoder accuracy:

$$CodeWiseAcc_min = \frac{n-t}{n}, \tag{3}$$

where $CodeWiseAcc_{min}$ is the lower bound of the required accuracy, n is the number of elements of the code sequence containing check and information symbols, t is the number of guaranteed error correction by the error code.

The boundaries of acceptable accuracy of the autoencoder when using anti-interference codes to provide lossless compression are presented in the Table 1.

Table 1. Limits of acceptable neural network accuracy

Interference-resistant code	Lower limit of tolerable accuracy of autoencoder
BCH (31, 21)	0,9355
BCH (31, 16)	0,9032
BCH (31, 5)	0,7580

3 Autoencoder Architecture

The neural autoencoder consists of the following layers:

1. The input layer that takes in a number of values equal to the length of the code and outputs a vector of length 16
2. Intermediate activation function

3. Hidden layer, taking a vector of length 16 at the input and giving out a vector of size equal to the length of the vector of the hidden state
4. Intermediate activation function

The neural decoder consists of the following layers:

1. The input layer that takes as input a number of values equal to the given length of the hidden state vector, and outputs a vector of length 31
2. Intermediate activation function
3. Hidden layer, taking a vector of length 31 at the input and outputting a vector of length 31 which outputs a vector of length 775
4. activation function
5. Hidden layer, taking a length vector of length 775 and which outputs a vector of size equal to the length of the code being used
6. Final activation function

The hyperbolic tangent was chosen as the intermediate activation function. A sigmoidal function was chosen as the final activation function.

For this study, three data sets were generated containing all possible code combinations of length equal to the number of information characters of the corresponding BCH code. Based on the information bits, noise-resistant code combinations were generated using the BCH coder. Since a complete data set fully describing all possible allowed code combinations is known, the division into validation and test part is not required. Binary Cross-entropy was chosen as the error function of the neural network. Training occurred until one of the conditions was met:

1. A metric value equal to one was achieved, indicating lossless data recovery
2. The neural network error value has stopped decreasing for a significant number of epochs, thereby the network has reached its recovery limit

By changing the hidden state of the neural network, there was a determination of the maximum compression ratio, at which the possibility of lossless data recovery was achieved using the appropriate noise-correcting code.

Determination of the possible advantage of using a noise-correcting code was performed by comparing the results with an autoencoder trained to recover only information bits, without the use of a noise-correcting code.

4 Experimental Results

4.1 BCH-Based Codec (31, 5)

During the study it was found that the use of a noise-correcting code made it possible to represent the initial combination in the form of a vector of length 2, while the model without the mechanism used showed itself worse, achieving lossless compression with a hidden state length equal to 3. The results are given in the Table 2.

Table 2. Results based on the BCH codec (31, 5)

Model reconstruction accuracy	The size of the hidden state vector		
Using anti-jamming code	1	2	3
False	0,78125	0,899999976	1
True	0,637499988	1	1

Thus, in the course of simulation with the use of interference-free coding the compression coefficient was achieved equal to 0.4. On the other hand, without the use of noise-correcting code compression factor is equal to 0.6. Thus, increase of code redundancy by 83.

The importance of the introduced metric $CodeWideAcc$ is clearly shown in Fig. 2. In the course of the experiment there was a contradiction: accuracy of the neural autoencoder lies in an acceptable zone, corresponding to lossless compression, but after the error correction algorithm the data recovery accuracy was 0.79 and 0.91, which does not correspond to the expected value of 1.

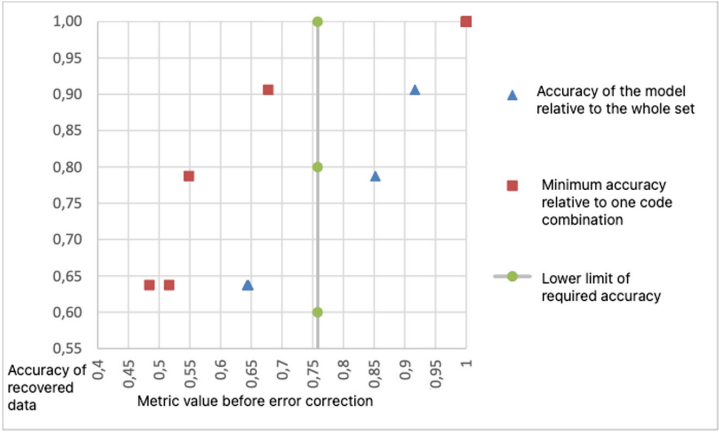

Fig. 2. Matching metrics before and after the error correction procedure

Figure 3 shows a graph of the change in the $CodeWiseAcc$ metric as a function of the number of training epochs with the hidden state equal to one. It shows that the accuracy of the approach without the use of interference-free code is higher over the whole interval, but it never reaches the value equal to unity. This difference between the metrics is due primarily to the fact that the neural autoencoder based on the noise-correcting code recovers a vector of length 31, while the model without the use of noise-correcting coding only 5 initial information bits.

Fig. 3. The value of the CodeWiseAcc metric when the length of the hidden vector is equal to one

Figure 4 shows a graph of the change in the value of the loss function depending on the number of epochs of training with the length of the hidden vector equal to one.

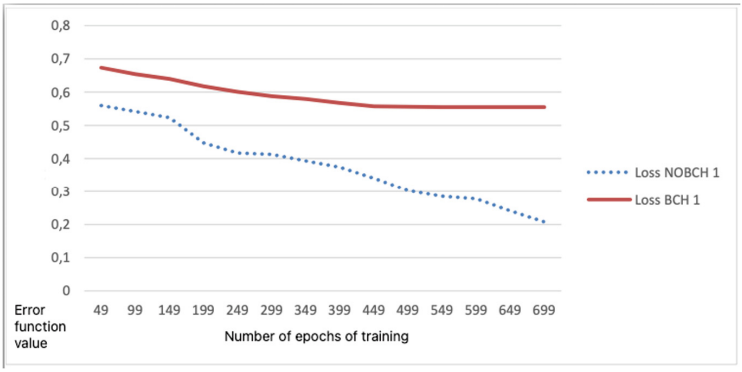

Fig. 4. The value of the loss function when the length of the hidden vector is equal to one

Figure 5 shows the graph of change in the metric $CodeWiseAcc$ depending on the number of epochs of training with the hidden state equal to two. It can be seen that starting from 649 epochs the neural network autoencoder, built using noise-free code, achieves accuracy equal to one, which indicates lossless data compression.

Figure 6 shows a graph of the change in the value of the loss function as a function of the number of training epochs with the length of the hidden vector equal to two.

Despite the fact that the neural network autoencoder using the noise-free coding algorithm achieved a lossless compression effect, the error function values

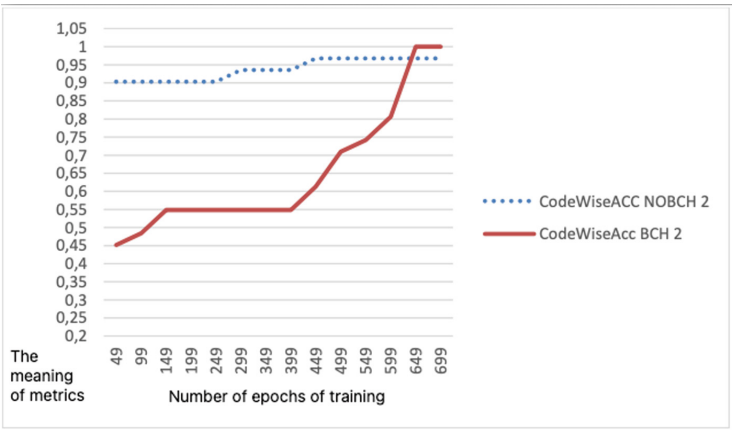

Fig. 5. The CodeWiseAcc metric value for a hidden vector length equal to two

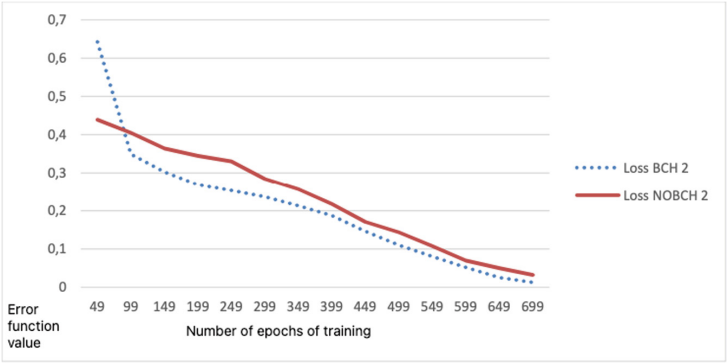

Fig. 6. The value of the loss function when the length of the hidden vector is equal to two

differ slightly, which confirms the theory of achieving full recovery of the original data by error correction.

4.2 BCH-Based Codec (31, 16)

During the study it was found that the use of a noise-correcting code made it possible to represent the initial combination in the form of a vector of length 7, while the model without the mechanism used showed itself worse, achieving lossless compression with a hidden state length equal to 8. The results are given in Table 3.

Table 3. Results based on the BCH codec (31, 16)

Model reconstruction accuracy	The size of the hidden state vector		
Using anti-jamming code	1	2	3
False	0,793	0,778	1
True	0,886	1	1

Thus, during the simulation with the use of interference-free coding the compression ratio was achieved equal to 0.43. On the other hand, without the use of noise-correcting code compression factor is equal to 0.5. Thus, a 52% increase in code redundancy entails a 12.5% decrease in the amount of data in the link.

Figure 7 shows the dependence of the obtained accuracy after data recovery by neural network autoencoder and the final accuracy after error correction. As can be seen, all the $CodeWiseAcc$ values lying on the interval included in the lossless compression interval have accuracy after error correction equal to one.

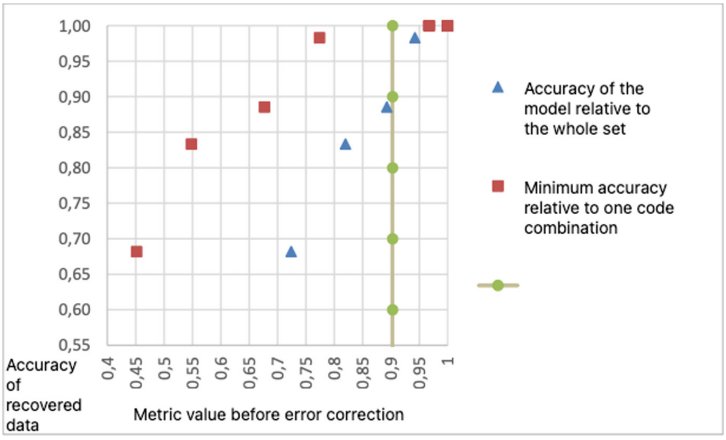

Fig. 7. Correspondence of metrics after code correction (31, 16)

Figure 8 shows the graph of change in the $CodeWiseAcc$ metric depending on the number of training epochs with the hidden state equal to 7. You can see that the accuracy is increasing despite regular drawdowns. The solution is to keep the intermediate model, which has the highest accuracy.

Figure 9 shows a graph of the change in the value of the loss function as a function of the number of training epochs with the length of the hidden vector equal to 7.

On this graph you can see a strong difference between the approaches. This fact is explained by the low accuracy of the model without the use of interference-resistant codes. Figure 10 shows a graph of the change in the $CodeWiseAcc$

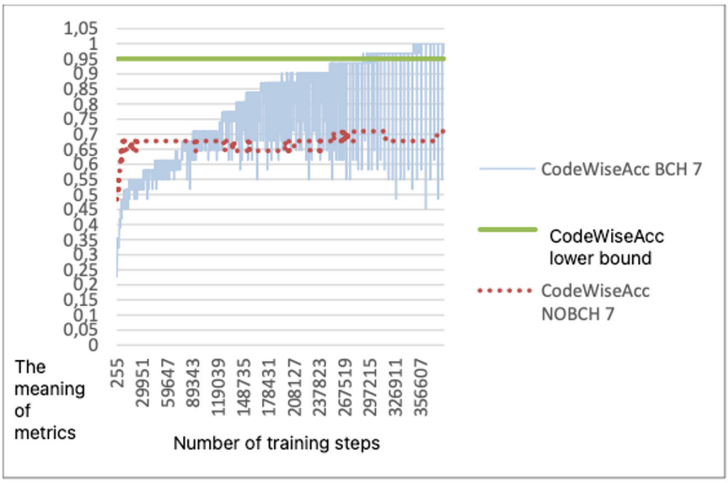

Fig. 8. The CodeWiseAcc metric value for a hidden vector length of seven

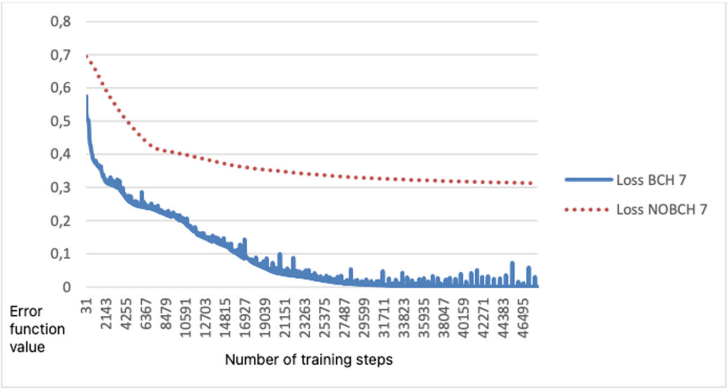

Fig. 9. The value of the loss function when the length of the hidden vector is equal to seven

metric as a function of the number of training epochs with the hidden state equal to 8.

Starting from 37567 epoch the neural network autoencoder constructed with the use of the interference-resistant code reaches the accuracy equal to one, which indicates lossless data compression. Neural network autoencoder without use of interference-free coding achieves this value at step 8863, which is much smaller. This fact is explained by the fact that the neural network autoencoder without the use of noise-correcting code recovers only 16 information symbols against 31.

Figure 11 shows a graph of the change in the value of the loss function depending on the number of training epochs with the length of the hidden vector equal to eight.

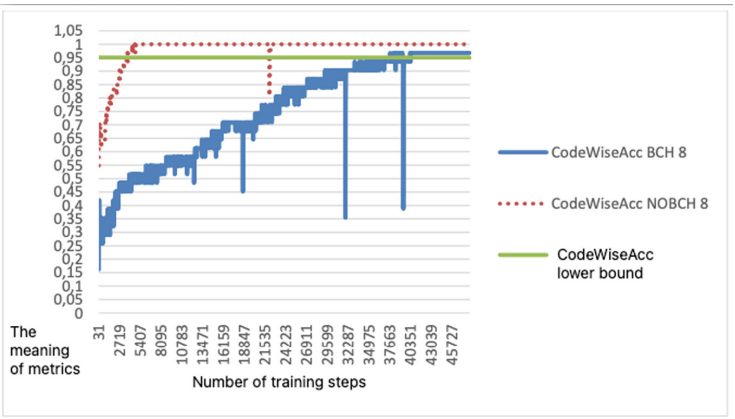

Fig. 10. The CodeWiseAcc metric value for a hidden vector length of eight

Fig. 11. The value of the loss function when the length of the hidden vector is equal to eight

Thus, the proposed implementation of the neural network codec allows to get rid of less redundancy in the channel while using the code with less correcting power. The reason of this fact may be the polynomial growth of the training set with the increase of information symbols, which requires to change the number of training parameters of the neural network in the higher side.

4.3 Code Based on BCH (31, 21)

During the study of the neural network autoencoder using the interference-resistant BSH code (31, 21), a contradiction was identified. A neural network autoencoder without the use of a noise-correcting code, having a hidden vector size equal to the length of the original sequence could not achieve the necessary accuracy for lossless compression. Table 4 shows the accuracy values depending on the hidden layer length and the condition of using the interference-resistant code.

Table 4. The results obtained based on the BCH codec (31, 21)

Model reconstruction accuracy	The size of the hidden state vector						
Using anti-jamming code	7	9	11	12	13	15	21
False	0,67	0,74	0,71	–	0,77	–	0,72
True	0,39	0,52	0,52	0,68	–	0,716	0,77

Figure 12 shows a graph of the dependence of the metric $CodeWiseAcc$ on the size of the hidden state of the neural network autoenumerator using the interference-resistant code. As can be seen from the graph during the experiment it was not possible to achieve the required accuracy, which guarantees lossless data compression. However, as shown in [3] it is possible to achieve this by controlling the power of the training set and filtering elements that do not affect the final generalization ability.

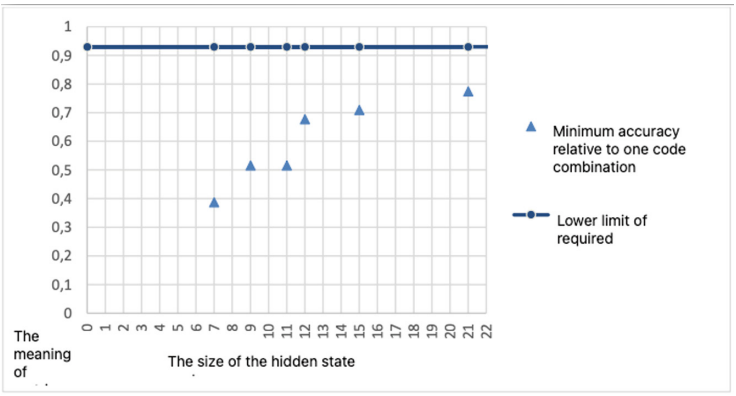

Fig. 12. CodeWiseAcc metric value depending on the hidden state

Figure 13 shows a graph of the change in the loss function value depending on the number of training epochs with the hidden vector length equal to 21

corresponding to the input combination length and using the neural network autoencoder model without using the noise coding mechanism. This graph shows that the values of the error function have reached a plateau, but the value of the function is not small enough to provide lossless compression.

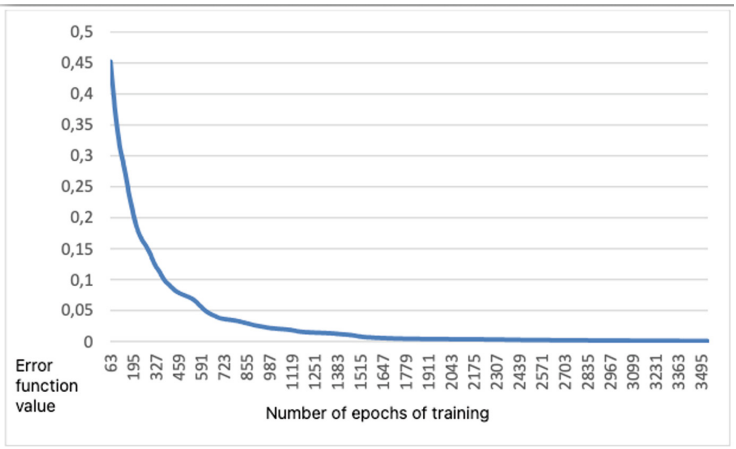

Fig. 13. The value of the loss function when the length of the hidden vector is equal to 21

Thus we can conclude that the structure of the neural network autoencoder, does not have sufficient complexity to provide lossless data compression for the given number of information symbols and the size of the training set. Therefore to study this code and codes with larger length of information bits it is necessary to use more complex architectures with a larger number of trainable parameters.

5 Conclusion

The following results were obtained during the analysis: using the FFT code (31, 5) it was possible to achieve a compression ratio equal to 0.4, which will reduce the data volume in the communication channel by 33% when modifying the architecture based only on the neural network autoencoder. When using the BCH code (31, 16), a compression ratio of 0.43 was achieved, which will reduce the amount of data in the communication channel by 12.5% when modifying the architecture based only on the neural network autoencoder. Also, the study revealed disadvantages that must be taken into account in the system design. During the comparison of the obtained accuracy when using codes with different minimum code distances, it was determined that the data structure formed by cyclic codes allows the neural network to better determine the weighting coefficients and restore the input sequence. The results of this comparison showed

that a larger minimum code distance of the input sequence leads to greater relative accuracy of the model. This means that when the minimum code distance increases, the recovery potential of the neural network autoencoder increases, which can be used to solve problems in related areas.

Acknowledgments. The study was financially supported by the Russian Science Foundation within of scientific project No. 22-49-02023 "Development and study of methods for obtaining the reliability of tethered high-altitude unmanned telecommunication platforms of a new generation".

References

1. Koucheryavy, A., Vladyko, A., Kirichek, R.: State of the art and research challenges for public flying ubiquitous sensor networks. In: Balandin, S., Andreev, S., Koucheryavy, Y. (eds.) ruSMART 2015. LNCS, vol. 9247, pp. 299–308. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23126-6_27
2. Wu, Y., et al.: Deep image compression with latent optimization and piece-wise quantization approximation. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, pp. 1926–1930 (2021). https://doi.org/10.1109/CVPR46437.2021
3. Berezkin, A., Zadorozhnyaya, A., Kukunin, D., Matveev, D., Kraeva, E.: Models and methods for decoding of error-correcting codes based on a neural network. In: 13th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), Brno, pp. 230–235 (2021). https://doi.org/10.1109/ICUMT54235.2021.9631637
4. Ahmed, S.: Linear block code decoder using neural network. In: IEEE International Joint Conference on Neural Networks (IJCNN 2008), Hong Kong, pp. 1111–1114 (2008). https://doi.org/10.1109/IJCNN13382.2008
5. Claytus Vaz, A., Nayak, G., Nayak, D.: Hamming code performance evaluation using artificial neural network decoder. In: 15th International Conference on Engineering of Modern Electric Systems (EMES), Oradea, pp. 37–40 (2019). https://doi.org/10.1109/EMES.2019.8795208
6. Al-Gaashani, M., A Muthanna, M.S., Abdukodir, K., Muthanna, A., Kirichek, R.: Intelligent system architecture for smart city and its applications based edge computing. In: 12th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), Brno, pp. 269–274 (2020). https://doi.org/10.1109/ICUMT51630.2020
7. Vladimirov, S., Kirichek, R., Vishnevsky, V.: Network coding for the interaction of unmanned flying platforms in data acquisition networks. In: The 4th International Conference on Future Networks and Distributed Systems (ICFNDS), St. Petersburg, pp. 1–7 (2020)
8. Vladimirov, S., Vishnevsky, V., Larionov, A., Kirichek, R.: The model of WBAN data acquisition network based on UFP. In: Vishnevskiy, V.M., Samouylov, K.E., Kozyrev, D.V. (eds.) DCCN 2020. LNCS, vol. 12563, pp. 220–231. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-66471-8_18
9. Vishnevsky, V.M., Tereschenko, B.N., Tumchenok, D.A., Shirvanyan, A.M., Sokolov, A.: Principles of building a power transmission system for tethered unmanned telecommunication platforms. In: Vishnevskiy, V.M., Samouylov, K.E., Kozyrev, D.V. (eds.) DCCN 2019. LNCS, vol. 11965, pp. 94–110. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36614-8_8

10. Zhang, M., Zhang, H., Yuan, D., Zhang, M.: Compressive sensing and autoencoder based compressed data aggregation for green IoT networks. In: Global Communications Conference (GLOBECOM), Waikoloa, pp. 11732–11742 (2019). https://doi.org/10.1109/GLOBECOM38437.2019.9013373
11. Zhou, Y., Wang, C., Zhou, X.: DCT-based color image compression algorithm using an efficient lossless encoder. In: 14th IEEE International Conference on Signal Processing (ICSP), Beijing, pp. 450–454 (2018). https://doi.org/10.1109/ICSP43694.2018
12. Xu, T., Darwazeh, I.: Design and prototyping of neural network compression for non-orthogonal IoT signals. In: Wireless Communications and Networking Conference (WCNC), Marrakesh, pp. 1–6 (2019). https://doi.org/10.1109/WCNC44850.2019
13. Yastrebova, A., Kirichek, R., Koucheryavy, Y., Borodin, A., Koucheryavy, A.: Future networks 2030: architecture & requirements. In: 10th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), Moscow, pp. 1–8 (2018). https://doi.org/10.1109/ICUMT45195.2018