UDC 004.075

INTERNET OF THINGS DATA TRANSFER METHOD USING NEURAL NETWORK AUTOENCODER

Eduard Siemens, Vasyl V. Kurdecha, Serhii M. Ushakov

Educational and Research Institute of Telecommunication Systems
Igor Sikorsky Kyiv Polytechnic Institute, Kyiv, Ukraine

Background. The number of devices in the Internet of Things is constantly increasing. At the same time, the number of solutions on the market for such technologies is growing. Statistics confirm that these factors lead to an increase in data transfer volumes. This raises the number of resources spent on data transmission. The growing trend in the number of users of the Internet of Things technology leads to the emergence of the problem of a rapid increase in the data transmitted by the network.

Objective. The purpose of the paper is to improve the process of data transmission in the Internet of Things by modifying the neural network autoencoder to reduce network resources use.

Methods. Analysis of publications dedicated to Internet of things data transmission. Integration of existing data coding solutions based on a neural network autoencoder in the process of transmitting data from the Internet of things.

Results. The neural network autoencoder has been improved by using an algorithm that additionally includes an arithmetic encoder and further training a new model on the output of a full-fledged autoencoder.

Conclusions. The process of data transmission in the Internet of Things network has been modified by improving the neural network autoencoder by using the training of a smaller neural network on the initial data of the main autoencoder, which has reduced the amount of data transmitted and, accordingly, reduced the use of network resources.

Keywords*: Internet of things; data coding; neural network; autoencoder; training on the original model; data compression.*

INTRODUCTION

The number of devices in the IoT network is constantly increasing. At the same time, the number of solutions in the IoT technology market is increasing, which in turn leads to an increase in data transfer volumes. This increases the amount of resources spent on ensuring their transfer.

The increase in the number of Internet of Things technology users is leading to a rapid increase in data transmitted over the network. The more traffic that reaches the network, the faster the data transfer rate. This complicates the process of exchanging traffic, which in turn leads to the need to spend resources on expanding network bandwidth.

PROBLEM FORMULATION

The main task in exchanging IoT data is to transfer it to a cloud environment based on short-range communication systems - personal networks. These can include both wireless and wired networks. The first ones include Bluetooth, NFC, RFID, Wi-Fi, Zigbee, Z-Wave protocols, while wired networks have a much wider list of technologies and names, as they include all possible industrial networks and protocols. At this stage, the main difficulty arises, which is the relative slowness of data transfer to the cloud and its further processing in the cloud. The main disadvantage of existing solutions is that they provide a low data compression ratio with a large number of sensors. At present, it is necessary to develop a data compression technique for IoT applications and devices with limited resources, efficiently working on multivariate time series and implemented on a real carrier.

As an option to address these shortcomings, first of all, it is necessary to apply a fast, error- and data-loss-aware compressor on the collected data before the transmitters, which is considered the largest traffic consumer in an IoT device. The second step is to recover the transmitted data at the edge node and process it using supervised machine learning methods.

The main aim of the paper is to improve the process of data transmission in the Internet of Things by modifying the neural network autoencoder to reduce the network resources use. The neural network autoencoder has been improved by using an algorithm that additionally includes an arithmetic encoder and further training a new model on the output of a full-fledged autoencoder.

USED TECHNOLOGIES.

Recurrent neural networks (RNNs) are a type of neural networks that excel in modelling sequence data like natural language or time series. RNNs have the unique ability to utilize their internal memory to process sequences of any length, unlike multilayer perceptrons. Consequently, they are suitable for tasks involving the segmentation of a whole into parts, for example, handwriting or language recognition. Recurrent networks have a wide range of architectural

solutions, from simple to complex. Presently, the most popular architectures are the long-term and short-term memory network (LSTM) and the gated recurrent unit (GRU).

To illustrate, the usage of RNNs can be represented schematically (refer to Fig.1): during each iteration of the loop, the model input receives both a fresh data fragment and the previous internal state. Eventually, the entire sequence is represented by the output of the neural network.

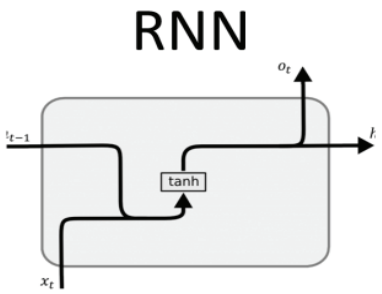

Fig. 1 Circuit diagram of a recurrent neural network

The Long Short-Term Memory (LSTM) is a deep learning architecture designed for artificial recurrent neural networks (RNNs) as illustrated in Figure 2. Unlike conventional neural networks, which use direct communication, LSTMs incorporate feedback with previous layers of the network. They can process entire data sequences such as speech or video, instead of individual data points like images. LSTMs have a wide range of applications, including non-segmented handwriting recognition, speech recognition, network traffic anomaly detection, and intrusion detection systems (IDS).

An LSTM unit comprises a cell, inlet valves, outlet valves, and forgetting valves. The cell stores values at irregular intervals, and the three valves regulate information flow into and out of the cell. LSTMs are suitable for processing, classification, and forecasting based on time-series data, as they can account for variable intervals between significant events in a sequence. They were designed to address the issue of the disappearing gradient that can occur during training of traditional RNNs. Hidden Markov models and other sequence learning techniques have difficulties with long-term dependencies in input sequences. In theory, classical RNNs can track arbitrary long-term dependencies in input sequences, but in practice, the computational process can cause inverse propagation gradients to vanish or explode due to finite accuracy.

LSTMs can solve the vanishing gradient problem by allowing gradients to flow unchanged. However, they can still experience the issue of exploding gradients. Nonetheless, LSTMs are relatively insensitive to interval length and can track long-term dependencies in input sequences. In conclusion, LSTMs are a powerful tool for processing data sequences and have broad applications in various fields, especially for problems involving time-series data with long-term dependencies.

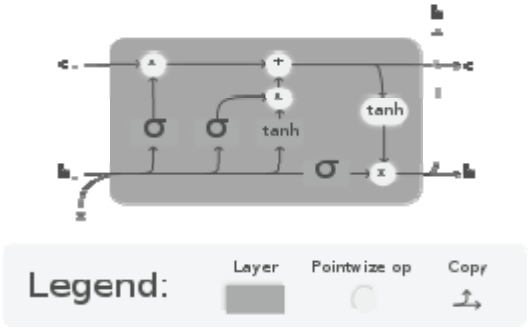

Fig. 2 LSTM circuit diagram

In 2014, Kyunghyun Cho et al. introduced gated recurrent units (GRUs) as a mechanism for constructing recurrent neural networks. The GRU is similar to long-term memory (LSTM) in that it has a forget mechanism, but it has fewer parameters than LSTM because it lacks an initial state. In tasks involving modelling polyphonic music, speech signals, and natural language processing, GRUs have demonstrated performance comparable to LSTM. Furthermore, on certain smaller and less frequent datasets, GRUs have demonstrated superior accuracy (Fig 3).

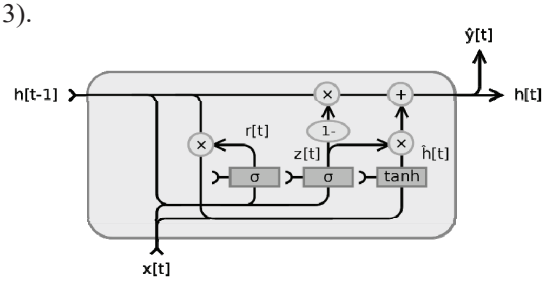

Fig. 3 The scheme of operation of the GRU circuit

DeepZIP is an innovative lossless compression technique that leverages a combination of Recurrent Neural Networks (RNN) and information theory techniques. At the heart of the DeepZip model lies the RNN for probability estimation, which estimates the likelihood of occurrence of each building block of the input sequence. The RNN receives the input symbol of the original sequence at each time step, which can be a bit or a byte, a base in the DNA chain, an English letter, or any other building block of the sequence. After processing the symbol, the RNN generates a vector that contains predictions of how likely each building block is to appear in the sequence. For instance, when encoding a sequence of bits, an output of [0.25, 0.75] would imply that the model assumes the next bit is 1

with a probability of 75%. Subsequently, the RNN shows the next character in the sequence for which it generates probabilities, given the characters that were shown to it earlier. One of the key differentiators of the RNN for probability estimation in DeepZip is that it does not require training before it processes new input. Unlike most neural networks, which use a training dataset to determine their internal parameters, the RNN for probability estimation starts with random parameters when it receives new input. As it processes symbols at the input, it updates its latent state according to the usual RNN rules, and it also updates its weight parameters using the loss between its probabilistic predictions and the actual symbol. Thus, the RNN for probability estimation does not just identify the dependencies that exist in the new sequence; it also learns how to study these dependencies. This approach provides an effective solution for lossless compression and improves the accuracy of the predicted probabilities. DeepZIP combines RNN with information theory techniques to create a highly efficient lossless compression technique. The RNN for probability estimation plays a key role in estimating the likelihood of occurrence of each building block of the input sequence, and its unique training methodology ensures that it can effectively identify and learn the dependencies in the new sequence.

The process of encoding data based on the probability estimates generated by the RNN requires a mechanism to translate these estimates into an actual encoded sequence. To achieve this, DeepZIP utilizes a well-established technique from information theory known as the arithmetic encoder. This encoder employs a numeric range to represent the input sequence, starting from an initial range of 0.0 to 1.0, which is then modified as follows (Fig.4).:

1) First, estimate the probability of the next character appearing in the input sequence using a statistical model, such as an RNN.
2) Divide the current range of the encoder into units, with each symbol having its own subdivision. The length of the sub-interval is proportional to the probability of the corresponding symbol obtained in step 1.
3) Read the next character from the input sequence.
4) Set the encoder's current range to the sub-interval of this symbol. The new range will be shorter or longer depending on the probability of the symbol.
5) If there are more characters in the input sequence, repeat from step 1. The encoder will keep updating its range for each character in the input sequence.

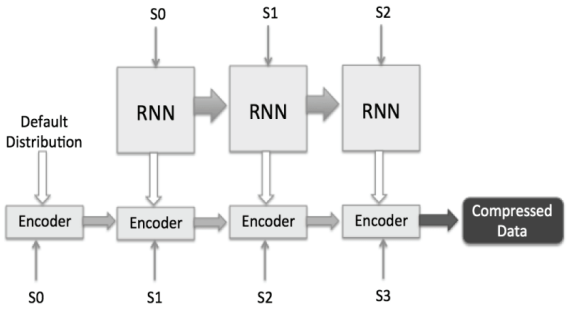

Fig. 4 Data compression scheme

To decode the compressed sequence, the arithmetic encoder is used in reverse order. Starting with the same range from 0.0 to 1.0, the decoder reads the compressed bit sequence one bit at a time. At each step, the decoder divides the current range into sub-intervals, one for each possible symbol, proportional to the probability of the corresponding symbol. The decoder selects the sub-interval that contains the binary fraction represented by the compressed sequence read so far. The symbol corresponding to that sub-interval is then output to the decompressed sequence. The decoder then updates the range to the sub-interval for the selected symbol and repeats the process until the entire compressed sequence has been decoded (Fig. 5):

1) Use the same model that was used to encode the message to predict the probability of occurrence of each character in the original sequence.
2) Divide the current range of the encoder into units, with one subdivision for each possible symbol, where the length of each subdivision is proportional to the probability of that symbol predicted in step 1.
3) Determine the unit that contains the coded number, which is a binary representation of a number from 0.0 to 1.0, contained in the final range of the arithmetic encoder at the encoding stage.
4) Add the symbol assigned to this subdivision in the original sequence. The final range of the encoding step was in the range selected for each previous step, so whatever subdivision of the character contains the input code, it must be in the output sequence.
5) Set the current encoder range for this unit. Now the range of the encoder is exactly the same as it was during the corresponding step of the encoding process.
6) Repeat steps 1-5 for each character to be decoded, until the end of the sequence is reached. The end of the sequence can be marked with a special character at the end of

the message or by knowing the number of characters that must be output. If this symbol is not the final symbol or the end of the sequence has not been reached, decoding continues.

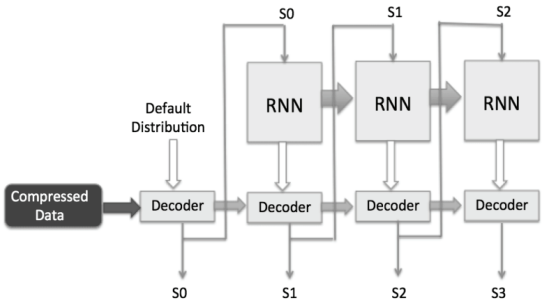

Fig. 5 Data decoding scheme

MAIN PART

In practical scenarios, the exact length of input sequences is generally unknown. To tackle this issue, an end-of-message character is often utilized to signal the encoder when to halt appending new characters to the output sequence. DeepZip, a method for encoding input sequences, leverages an RNN for probability estimation and an arithmetic encoder. Initially, the RNN is initialized with random weights to predict the probability. Next, the arithmetic encoder encodes the first character of the input sequence using the default character distribution. The first symbol is then fed to the RNN for probability estimation, which produces probabilities for the subsequent symbol. These probabilities are used by the arithmetic encoder to encode the second character. The RNN's weights for estimating the probability are updated by comparing the predicted probabilities for the second symbol with the actual identity of the second symbol. The second character is then introduced to the RNN to estimate the probability, which gives the probability for the third character, and the process continues until the input sequence is fully encoded.

In the context of encoding input data, it is noteworthy that the RNN used for probability estimation in the decoding process is initialized with the same weights that were employed at the outset of coding. One effective approach to achieve this is by transmitting a random value that was utilized during encoding, alongside the actual encoding of the input data. Subsequently, the arithmetic encoder adopts the default character allocation to extract the first character from the encoded data. This symbol is transmitted to the RNN to estimate the probability, which yields a set of probabilities for the next symbol. If properly initialized, these probabilities will correspond exactly to those issued by the RNN during the initial encoding step. The probabilities are then utilized by the encoder to extract the second character, which in turn is used to restore the RNN weights. The updating of scales should precisely mirror the process that occurred after the first coding step. The second character is then passed to the network, which issues probabilities for the third character, and the process is repeated until the encoder reads the end-of-message character.

PROPOSED METHOD DEFINITION

In the previous section, the compression algorithm using DeepZip neural networks was considered. However, in its development, the main platform was considered classical computer systems, such as web servers and personal computers. For use in IoT devices, the model may be too heavy: both in terms of memory required and in terms of consumption of computing resources. This is not the first time that the problem of heavy models has arisen in the field of neural networks. An example is BERT, the neural network architecture from Google, which allows you to recognize the meaning of quite complex sentences and even extract from them the semantic meanings of individual phrases. In itself, BERT training takes a very long time, resulting in a fairly large network, which could not be used to analyse all search queries. On the other hand, due to technical and architectural limitations, it is not possible to qualitatively teach simpler architecture at once. Thus, it is possible to teach a large model, but due to its size, the latter cannot be applied in practice. For this reason, the following principle is applied: first, the three-dimensional model is trained to an acceptable level, and only then there is training, compression and a light version of the model on the original model. Simply put, we do not try to immediately learn to answer questions about the text, but first learn to understand its meaning by observing how it is understood by the parent model, and learning to repeat it (Fig.6).

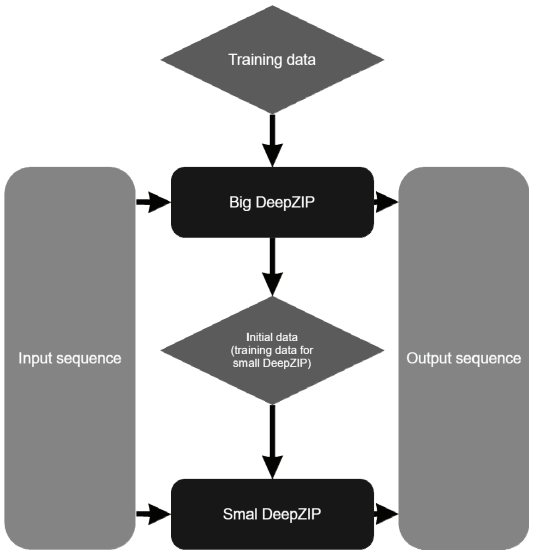

Fig. 6 Visualize the sequence of the data encoding method (diamond– model training, Superellipse– model testing)

EMPIRICAL MODELLING AND ANALYTICAL EVALUATION OF NEURAL NETWORKS

It was decided to abandon the training in parallel coding in favour of a pre-trained model, which will first put in the model features of the type of data transmitted, and secondly apply the above principle of compression by a recurrent neural network. Thus, the first step will be to train the RNN on the generated data set to be transmitted. In this way a neural network will be obtained, with some level of redundancy, which will be further eliminated by means of compressed to lower dimensions. It was decided to conduct an experiment using the most common application format JSON. It allows you to set quite complex data structures, while being quite simple and minimalist as opposed to cumbersome XML.

SIMULATION

Its features will be used for efficient data compression. To do this, a utility was developed to generate input data. Generation is done using the recursive function generate (size_left, max_depth), which returns a random object, which when serialized in JSON will give a string no longer than size_left bytes and a depth of no more than max_depth. At each step of the recursion, we try to add random keys to the object, partly from the dictionary, for which the values will be either random strings, or other random objects, or arrays of random objects (Fig.7).

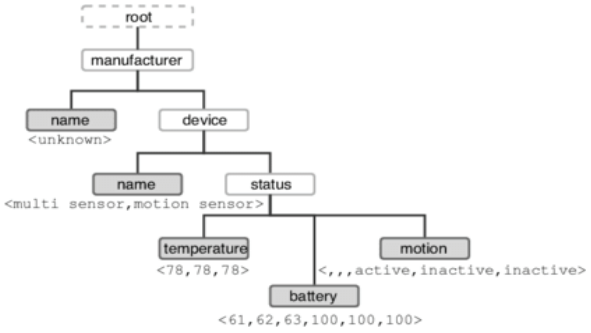

Fig. 7 JSON structure corresponding to one of the recursive call variations.

The original DeepZIP architecture, based on learning parallel to the compression process itself. Thus, for compression using this algorithm, it is necessary to update the scales after each portion of input data, i.e. the model learns the features of the data as it passes through them. In our case, the work is with the JSON format with a pre-known set of keys, which allows you to lay down knowledge about the structure of input data before the network will be used in practice. Thus, in the course of this work, a large neural network was trained, and then, based on a large one, a simpler model was trained, which can then be used on target IoT devices. Therefore, a set of randomly generated JSON-string algorithms with IoT-like data will be used for training. 200,000 rows up to 10 (approximately 2 MB of data), 20,000 rows up to 50, 10,000 rows up to 100, and 5,000 rows up to 10,000 were generated. This sample was used to teach large network architecture (Fig.8).

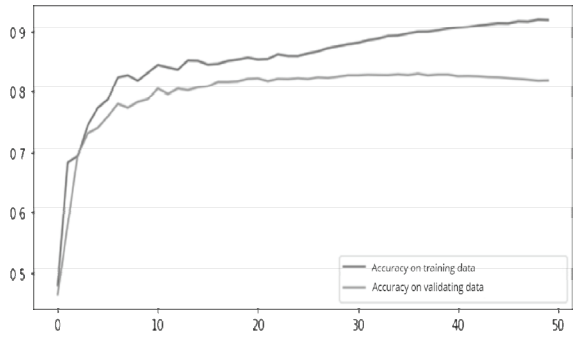

Fig. 8 Comparison of big network accuracy

A part of this sample will be used later to train the small model, as well as an additional random set of JSON strings will be generated. Thus, 50% of the sample of large architecture is used to teach small architecture, as well as the remaining 50%, regenerated to avoid retraining

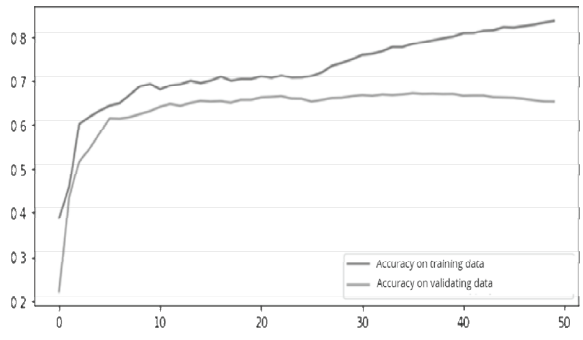

Fig. 9 Comparison of the accuracy of the new network

At the same time, given the differences in architectures, the small one is 3 times faster. The graphs above show the differences in the degree of compression depending on the number of eras for large and small architectures. The simplification of the DeepZIP architecture has provided a greater degree of compression specifically for IoT-specific data. Thus,

the smaller architecture trained on final weights of a big network gave approximately 12% less degree of compression at acceleration in 3 times (Fig.9).

CONCLUSION

The Internet of Things as a concept and a network is anticipated to integrate advanced technologies in the fields of telecommunications, cloud computing, and fog computing, as well as sensing, thus opening the door to groundbreaking applications in various areas that will bring many benefits and have a significant impact on people's lives. However, the nature of the IoT network, with its massive number of connected devices and large volumes of potentially vulnerable data, raises serious concerns regarding bandwidth, security, privacy, and performance. The solutions discussed in the paper represent an important step towards overcoming these challenges, but further analysis and comparison are needed to determine the advantages of implementing these systems or their combinations on various IoT network infrastructures, both current and future. Although there is no "one-for-all" solution that can be easily created, it is still possible to develop new and improved approaches and systems to enhance the security of the IoT network. One possible solution is to integrate existing data transmission methods and combine them with modern technologies from neighbouring fields, such as AI-based compression, as demonstrated in the article or other methods.

REFERENCES

1. Mashal, I.; Alsaryrah, O.; Chung, T.Y.; Yang, C.Z.; Kuo, W.H.; Agrawal, D.P. Choices for interaction with things on Internet and underlying issues. Ad Hoc Netw. 2015.

2. Madakam, S.; Ramaswamy, R.; Tripathi, S. Internet of Things (IoT): A literature review. J. Comput. Commun. 2015.

3. Sethi, P.; Sarangi, S.R. Internet of Things: Architectures, Protocols, and Applications. J. Electr. Comput. Eng. 2017.

4. M. Goyal, K. Tatwawadi, S. Chandak, I. Ochoa, Data Compression Conference (DCC), May 2019

5. Compressing BERT for faster prediction // Retrieved from https://blog.rasa.com/compressing-bert-for-faster-prediction-2/

6. Convert the sample JSON file from a tree to a table. // Retrieved from https://www.researchgate.net/figure/Convert-the-sample-JSON-file-from-a-tree-to-a-table_fig3_332591327

7. NN based lossless compression – DeepZip // Retrieved from https://github.com/mohit1997/DeepZip

8. D.D. Testa and M. Rossi, IEEE Signal Processing Letters, 2015.

9. L. Globa, V. Kurdecha, I. Ishchenko and A. Zakharchuk, "An approach to the Internet of Things system with nomadic units developing," 2017 14th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM), 2017, pp. 248-250, doi: 10.1109/CADSM.2017.7916127.

10. J. Yamnenko, L. Globa, V. Kurdecha and A. Zakharchuk, "Data Processing in IoT Systems based on Fuzzy Logics," 2019 Modern Electric Power Systems (MEPS), 2019, pp. 1-4, doi: 10.1109/MEPS46793.2019.9395055.

11. J. Yamnenko, V. Kurdecha and N. Gvozdetska, "Domestic Solid Waste Disposal Logistic Optimization Using Internet of Things Technologies," 2021 IEEE International Conference on Information and Telecommunication Technologies and Radio Electronics (UkrMiCo), 2021, pp. 1-5, doi: 10.1109/UkrMiCo52950.2021.9716596.

12. L. Globa, V. Kurdecha, I. Ishchenko, A. Zakharchuk and N. Kunieva, "The Intellectual IoT-System for Monitoring the Base Station Quality of Service," 2018 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom), 2018, pp. 1-5, doi: 10.1109/BlackSeaCom.2018.8433715

13. Skulysh M. Management of Multiple Stage Queuing Systems / M. Skulysh, S. Sulima // CADSM 2015 : 13-th International conference, 24–27 February 2015 : conference proceedings. — Lviv–Polyana, 2015. — pp. 431– 434.

14. G. Amrani, A. Adadi, M. Berrada, Z. Souirti and S. Boujraf, "EEG signal analysis using deep learning: A systematic literature review," 2021 Fifth International Conference On Intelligent Computing in Data Sciences (ICDS), 2021, pp. 1-8, doi: 10.1109/ICDS53782.2021.9626707.

15. I. V. Pustokhina, D. A. Pustokhin, D. Gupta, A. Khanna, K. Shankar and G. N. Nguyen, "An Effective Training Scheme for Deep Neural Network in Edge Computing Enabled Internet of Medical Things (IoMT) Systems," in IEEE Access, vol. 8, pp. 107112-107123, 2020, doi: 10.1109/ACCESS.2020.3000322.

16. S. Huang, Y. Guo, D. Liu, S. Zha and W. Fang, "A Two-Stage Transfer Learning-Based Deep Learning Approach for Production Progress Prediction in IoT-Enabled Manufacturing," in IEEE Internet of Things Journal, vol. 6, no. 6, pp. 10627-10638, Dec. 2019, doi: 10.1109/JIOT.2019.2940131..

17. M. U. Ndubuaku, M. K. Ali, A. Anjum, A. Liotta and S. Reiff-Marganiec, "Edge-enhanced analytics via latent space dimensionality reduction," 2020 IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT), 2020, pp. 87-95, doi: 10.1109/BDCAT50828.2020.00018.

18. Liu, C., Zhang, Y., Li, Z., Zhang, J., Qin, H., & Zeng, J. (2015). Dynamic Defense Architecture for the Security of the Internet of Things. 2015 11th International Conference on Computational Intelligence and Security (CIS). doi:10.1109/cis.2015.100

Сіменс Е., Курдеча В.В., Ушаков С.М.

Метод передачі даних мережі Інтернету речей з застосуванням нейромережевого автоенкодеру

Проблематика. Кількість пристроїв в мережах Інтернету речей постійно збільшується. Разом з цим збільшується кількість рішень на ринку таких технологій. Статистика підтверджує, що дані чинники призводить до зростання об'ємів передачі даних. Тим самим підвищується кількість ресурсів, що витрачається на забезпечення передачі даних. Тенденція зростання кількості користувачів технології Інтернету речей призводить до появи проблеми стрімкого збільшення даних, що передаються мережею.

Мета дослідження. Удосконалити процес передачі даних в мережі Інтернету речей за рахунок модифікації нейромережевого автоенкодеру для зменшення використання ресурсів мережі.

Метод реалізації. Аналіз публікацій, присвячених передачі даних мережі Інтернет речей. Інтеграція існуючих рішень кодування даних на основі нейромережевого автоенкодеру в процесі передачі даних мережі Інтернету речей.

Результати. Удосконалено нейромережевий автоенкодер за рахунок використання алгоритму, що додатково включає в себе арифметичний кодер та подальшого навчання нової моделі на вихідні данні повноцінного автоенкодеру.

Висновки. Модифіковано процес передачі даних в мережі Інтернету речей за рахунок удосконалення нейромережевого автоенкодеру за допомогою використання навчання меншої нейромережі на вихідних даних основного автоенкодеру, що дозволило зменшити кількість даних, що передаються і відповідно зменшення використання ресурсів мережі.

Ключові слова: *Інтернет речей; кодування даних; нейронна мережа; автоенкодер; навчання на вихідну модель; стиснення даних.*