Implementation of probabilistic data structures in the processes of neuroevolutionary synthesis

Serhii Leoshchenko[a], Andrii Oliinyk[a], Sergey Subbotin[a], Viktor Lytvyn[a] and Oleksandr Korniienko[a]

[a] *National university "Zaporizhzhia polytechnic", Zhukovskogo street 64, Zaporizhzhia, 69063, Ukraine*

Abstract
Neuroevolutionary synthesis is an alternative to training a pre-designed neural network. The use of neuroevolution methods is explained by fewer free parameters, less dependence on the expert, and usually better adaptive characteristics. However, neuroevolution methods are more resource-intensive, because the synthesis uses populations consisting of a large number of networks. Various mechanisms, such as parallelization, can be used to solve problems related to synthesis time. However, they usually lead to even more memory usage in the process. Mechanisms such as selective pressure only partially solve this problem. Probabilistic data structures have proven to be a fairly compact storage mechanism. Therefore, the paper provides an example and analysis of ways to use probabilistic data structures during the neuroevolutionary synthesis of artificial neural networks.

Keywords 1
Neuroevolutionary, synthesis, artificial neural networks, probabilistic data structures, encoding, sequencing, genetic operators

1. Introduction

Artificial neural networks (ANN) are one of the areas of scientific research in the field of creating artificial intelligence [1-3], which is based on the desire to imitate the human nervous system [3-5]. Including its (nervous system's) ability to correct mistakes and self-learn [5-7]. The scope of application of ANNs is constantly expanding, today they are used in such areas as:

- machine learning, which is a type of artificial intelligence [1-7]. It is based on training the ANN on the example of millions [8-11] of tasks of the same type. Nowadays, machine learning is actively implemented by Google [12-15], Bing [16], [17], and Baidu [18], [19] search engines. So based on the millions of search queries that we all enter into search systems every day, their algorithms learn to show us the most relevant results so that we can find exactly what we are looking for [14], [15];
- in robotics [20], [21], neural networks are used to develop numerous algorithms for the Iron "brains" of robots. Here, training usually uses methods that implement a reinforcement learning strategy, such as swarm intelligence or the actor-critical method (A2C and A3C) [22-25];
- computer system architects use neural networks to solve the problem of parallel computing [4], [7];
- with the help of neural networks, scientists can solve various complex mathematical problems [6].

As already mentioned in one of the theses, before direct training of the ANN, it is necessary to train. This can be done using certain examples (supervised learning), using information about the environment and the state of the "agent" in that environment (reinforcement learning), or without any

CMIS-2021: The Fourth International Workshop on Computer Modeling and Intelligent Systems, April 27, 2021, Zaporizhzhia, Ukraine
EMAIL: sergleo.zntu@gmail.com (S. Leoshchenko); olejnikaa@gmail.com (A. Oliinyk); subbotin@zntu.edu.ua (S. Subbotin); lytvynviktor.a@gmail.com (V. Lytvyn); al.korn95@gmail.com (O. Korniienko)
ORCID: 0000-0001-5099-5518 (S. Leoshchenko); 0000-0002-6740-6078 (A. Oliinyk); 0000-0001-5814-8268 (S. Subbotin); 0000-0003-4061-4755 (V. Lytvyn); 0000-0003-4812-5382 (O. Korniienko)

clear input information at all (unsupervised learning) [26-33]. Neuroevolution methods can be used for all approaches, because usually, in addition to conventional training (parametric synthesis), neuroevolution methods adjust the structure (topology) of the ANN (structural synthesis) [37], [38]. Such methods are separated into a separate group of methods for the evolution of interneuronal connections and network topologies (TWEANNs) [37] ,[38], which are individuals in the population.

However, one of the main disadvantages of neuroevolution methods is their resource intensity. Large amounts of synthesis time are usually explained by the search for the most optimal ANN structure, which is achieved by using evolutionary mechanisms in individual populations. On the other hand, taking into account the fact that in usual network training, using gradient methods, it is necessary to design topologies first and only then train it, it should be noted that neuroevolutionary synthesis automates this subprocess [39-42]. Moreover, the use of parallelization can significantly speed up the synthesis process [39-42]. And given the fact that almost all neuroevolution methods can be parallelized, this removes the speed problem. However, parallelization imposes additional memory requirements on the computing system. The use of selective pressure mechanisms [43] can increase the level of adaptability of final decisions and reduce memory usage, but in sequential systems it can lead to a low level of genetic diversity.

Therefore, the question arises about using certain special structures to preserve information about the population. Moreover, the use of these structures is possible at the stage of encoding genetic information about an individual, even at the beginning of synthesis.

Probabilistic data structures (PDS) are a group of data structures that are extremely useful for big data and streaming applications [44-46]. Generally speaking, these data structures use hash functions to randomize and compactly represent a set of elements. Data collisions are ignored (usually using a number of mechanisms that prevent such situations), and errors are controlled at a certain threshold. Compared to error-free approaches, these algorithms use much less memory and have a constant query time. Moreover, support for union and intersection operations provides a low level of parallelization complexity.

Therefore, let's look at what stages of synthesis the use of PDS is justified and appropriate. The main stages are shown in Figure 1.

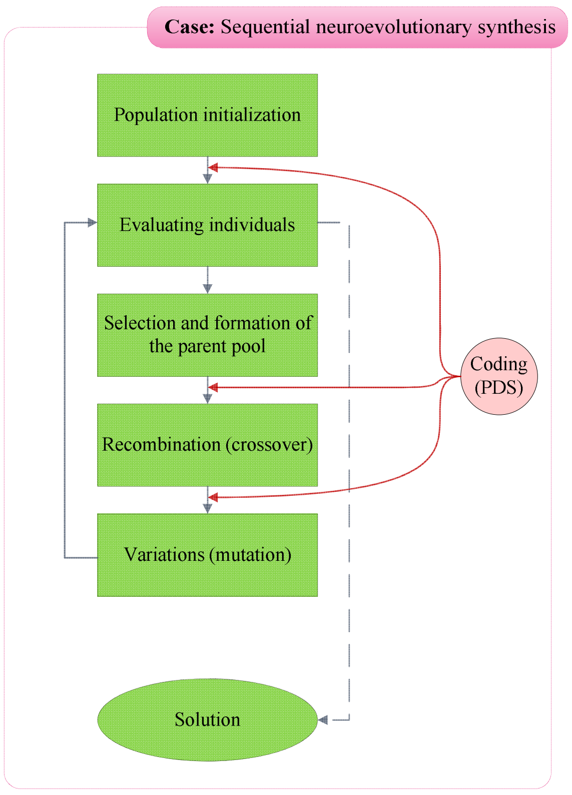
Case: Sequential neuroevolutionary synthesis
Population initialization
Evaluating individuals
Selection and formation of the parent pool
Recombination (crossover)
Variations (mutation)
Coding (PDS)
Solution

Figure 1: Synthesis stages where PDS can be implemented

Analyzing Figure1 note that the implementation of PDS during synthesis is advisable at the following stages:

- encoding of genetic information about ANN, as individuals of the population;
- crossover of individuals (since PDS supports unification and intersection).

Special cases can be considered situations when information about the ANN is transmitted, for example, between the cores of parallel computing systems or computing nodes, as shown in Figure 2.

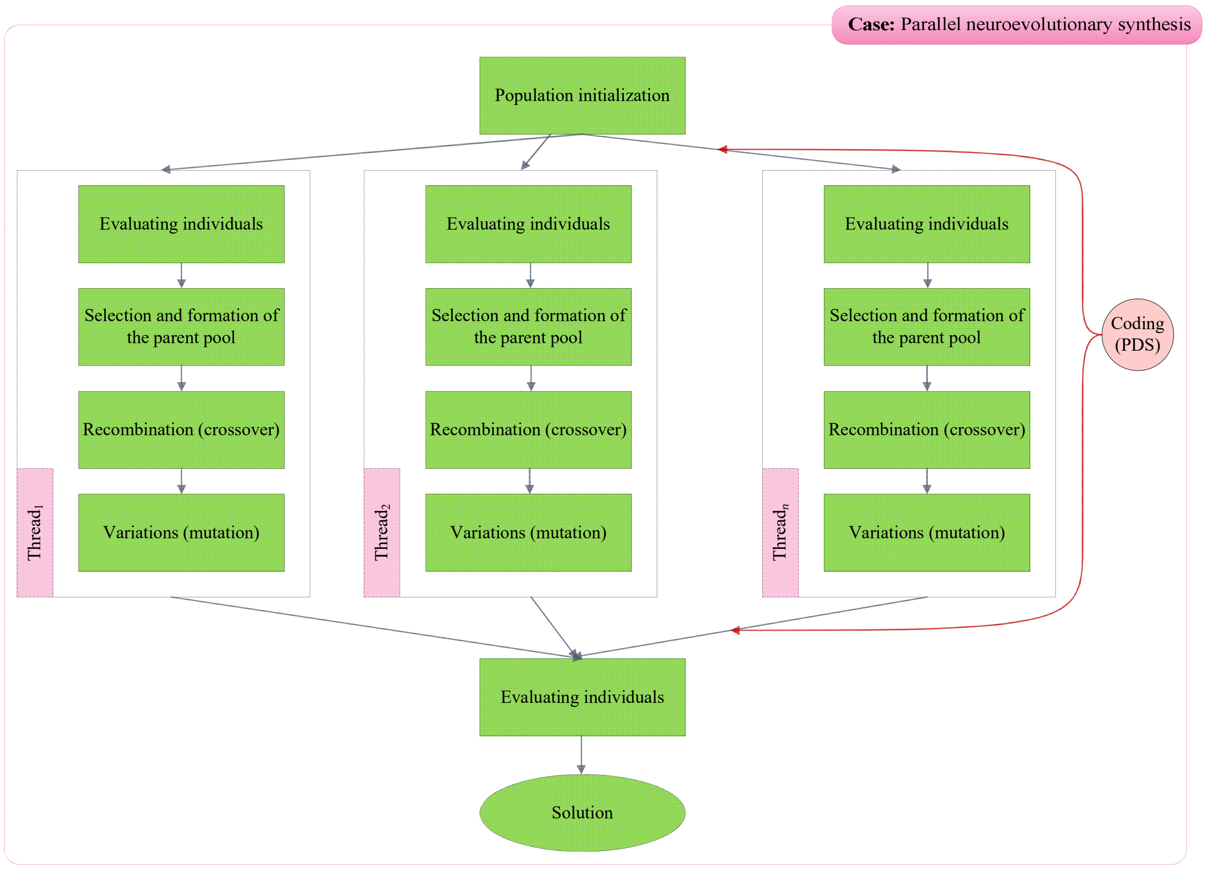
Case: Parallel neuroevolutionary synthesis
Population initialization
Evaluating individuals
Selection and formation of the parent pool
Recombination (crossover)
Variations (mutation)
Thread$_1$
Thread$_2$
Thread$_n$
Coding (PDS)
Evaluating individuals
Solution

Figure 2: ANNs forwarding in parallel systems

2. Literature Review

Statistical analysis and analysis of data that related to big data is a common and quite relevant task, especially in such areas as smart systems, Internet of things (IoT), Cognitive Internet of Things (CIoT), web analytics [44-47]. One of the options for organizing data storage when analyzing big data is powerful distributed databases and data warehouses: Hydra [48], [49], Hadoop [50] and NetApp [51], [52]. For processing, the same methods can be used, as in MapReduce, to search for a HashSet [49], [52]. However, such approaches are fraught with low problems in real-time data processing, due to the excessive importance of high-latency analytical processes and poor applicability. Therefore, if simple additive metrics (such as total transactions, total page views, or average conversion price) are most valuable during analytics, it is obvious that the raw data can be effectively summed up, for example, on a daily basis or using simple counters in the flow. Calculating more complex metrics (such as average humidity, the number of unique users, or the most frequently encountered elements) is more complex and requires more resources if implemented consistently. PDS allows to evaluate these and many other metrics and manipulate [44-46] the accuracy of estimates for memory consumption. These data structures can be used as temporary data accumulators in query processing procedures, or perhaps more importantly, as a compact, sometimes surprisingly compact, replacement for raw data in streaming computing.

A number of PDS's will be analyzed in abstracts, while others are described in detail using a detailed mathematical analysis of these structures can be found in the original articles. Preliminary comments are as follows [47]:

- for some structures, such as Loglog Counter or Bloom filter, there are simple and practical formulas that allow you to determine the structure parameters based on the expected amount of data and the required error probability. Other structures, such as Count-Min Sketch or Stream-Summary, have a complex dependence on the statistical properties of data, and experiments are the only reasonable way to understand their applicability to real-world use cases;
- the applicability of the PDSs is not strictly limited to the above queries or a single set of data. Experiments and other works prove that structures filled with different data sets can often be combined to handle complex queries, and other types of queries can be supported using custom versions of the described algorithms.

2.1. Bloom filter

Bloom filter is probably the most well-known and widely used PDS [53-56]. Bloom filter is similar to Linear Counting, but it is designed to maintain the identity of each element, not statistics. A Bloom filter is a bit array of M bits initialized at 0. To add an element, it is processed by k hash functions to get k array positions, and set the bits in these positions to 1. To request an element, it is passed to k hash functions to get k array position. If any of the bits in these positions are 0, then the element is definitely not included in the set. If all bits are equal to 1, then the element can be in the set. That is, If the filter has a relatively large size compared to the number of individual elements, each element has a relatively unique signature and you can check a specific value – whether it is already registered in the bit set or not. If all the bits of the corresponding signature are units, then the answer is yes (with a certain probability, of course). Bloom filter with a false positive rate of 1% requires only 9.6 bits per element, regardless of the size of the elements [54-56].

Similar to Linear Counting, Bloom filter supports bit typing. However, as the from description makes clear, each value is mapped not to one, but to a certain fixed number of bits using several independent k hash functions.

Let's consider formulas that allow us to calculate the parameters of the Bloom filter as a function of error probability and capacitance. Given false positive probability p and the estimated number of insertions n, the length of the bit array can be calculated as [54-56]:

$$m = -\frac{n \ln p}{\ln^2 2}, \tag{1}$$

where m is filter (sketch) size (bits);

p is an error probability (false positive);

n is maximum cardinality (capacity).

The hash functions used for bloom filter should generally be faster than cryptographic hash algorithms with good distribution and collision resistance.

$$k = \frac{m}{n} \ln 2, \tag{2}$$

where k is number of hash functions.

In Figure 3 shows an example of placing elements in Bloom filter.

Consider an example: we enter x, y and z in the filter with $k = 3$ hash functions, as shown in the figure above. Each of these three elements has three bits, each with a value of 1 in the bit array. When we search for w in the set because one of the bits is not set to 1, Bloom filter will tell us that it is not in the set [53-56].

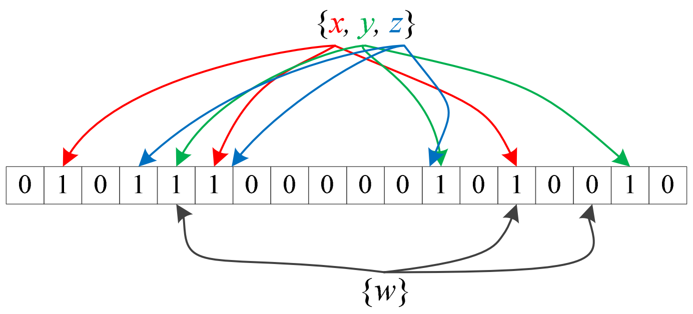

Figure 3: Placing elements in Bloom filter

2.2. MinHash

To begin with, MinHash is not just a PDS. In computer science and data mining, MinHash is a method for quickly estimating how similar two sets are. This scheme was invented by Andrei Broder (1997) [57] and was originally used in the AltaVista search engine to detect duplicate web pages and exclude them from search results [58]. It has also been used in large-scale clustering tasks, such as clustering documents based on the similarity of their word sets.

Let's introduce the concept of Jacquard similarity coefficient (Jaccard similarity coefficient) [57], [58]:

$$J[A,B]=\frac{|A\cap B|}{|A\cup B|}. \tag{3}$$

In other words, the number of elements in the cross-section is divided by the number of elements in the union. This estimate is called the Jacquard coefficient, the coefficient is zero when the sets have no common elements, and one when the sets are equal, in other cases the value is somewhere in the middle. In general, the calculation of such a coefficient boils down to the fact that at the beginning we need to divide the sets into separate parts, these will be the elements of our sets, then we need to somehow calculate the dimensions of the intersection and union. Usually, to efficiently perform the last two operations, sets are represented as hash tables without key-associated values.this structure works very quickly. Building a table: $O(n)$, you need two of them, calculating the cross section: $O(n)$ and calculating the union is also $O(n)$, where n is the number of elements in the set [57], [58].

Here are two sets A, B and h a hash function that can count hashes for elements of these sets. Next, we define the function $h_{\min}(S)$, which calculates the function h for all members of any set S and returns its lowest value. Now, let's start calculating $h_{\min}(A)$ and $h_{\min}(B)$ for different pairs of sets, the question is: what is the probability that $h_{\min}(A)=h_{\min}(B)$.

This probability must be proportional to the size of the intersection of sets: in the absence of common terms, it tends to zero, and to one, when the sets are equal, in intermediate cases it is somewhere in the middle. This is the $J(A,B)$ so Jacquard coefficient.

However, if we just count $h_{\min}(A)$ and $h_{\min}(B)$ for our two sets and then compare the values, it won't give us anything, because there are only two options: equal or not equal. We need to somehow get enough statistics for our sets to calculate $J(A,B)$ close to the truth. This is done simply, instead of one function h, we introduce several independent hash functions, or rather k functions [57]:

$$k=\left[\frac{1}{\varepsilon^2}\right], \tag{3}$$

where ε is the highest error value is desired. So, to calculate $J(A,B)$ with an error of no more than 0.1, you need 100 hash functions. On the one hand, this is a lot. So let discuss in what will be the optimization.

First, we can calculate the so-called $H_{\min}(S)$ signature, i.e. the minima of all hash functions for the set S. The complexity of calculating it is greater than when building a hash table, but, nevertheless, it is linear, and you only need to do this for each document once, for example, when adding it to the database, then it is enough to operate only with signatures [58].

Secondly, the signature has a fixed size for a given maximum error value. In order to compare any two sets of any size, you need to perform a fixed number of operations. In addition, in theory, signatures are much more convenient to index. In theory, because their indexing does not fit very well on relational databases, it is also not particularly suitable for full-text engines.

2.3. Count-Min Sketch

Count-min sketch is a PDS that is a table of event frequencies in a data stream. It uses hash functions to map events to frequencies, but unlike a hash table, it uses only sublinear space, by recalculating some events due to collisions [47], [59].

The Count-min sketch structure can be represented as a sketch with depth d and width w as at Figure 4.

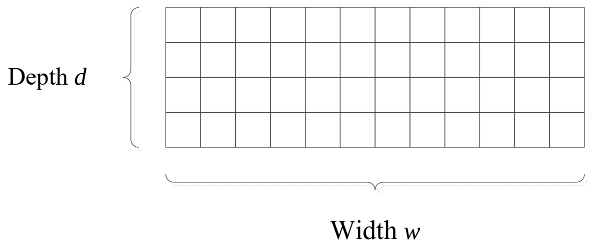

Figure 4: Count-min sketch with depth *d* and width *w*

Then we represent the distribution of values in the Count–min sketch array, based on the principle shown in Figure 5 this distribution can be considered a sketch.

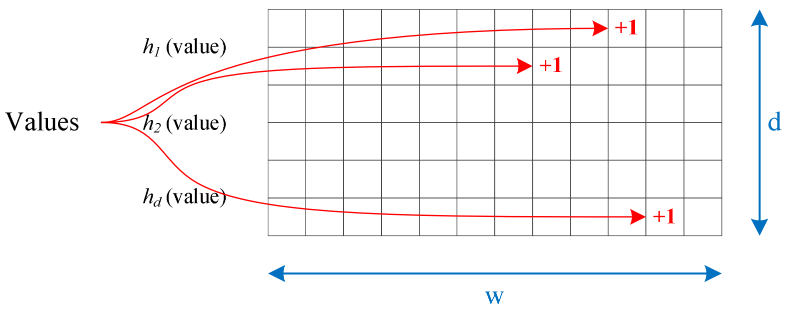

Figure 5: Distribution of values in the structure

In general, we will have a representation with a distribution according to the following rules (4) and (5).

$$\varepsilon_{estimation} \le 2 \cdot \max cardinality_{hashfunction} / width_{sketch}, \quad (4)$$

$$\delta = 1 - 0.5^{depth_{sketch}}, \quad (5)$$

$\varepsilon_{estimation}$ is an evaluation error;

δ is a probability for a value with Count-min sketch;

$cardinality_{hashfunction}$ is a power of the hash function;

$depth_{sketch}$ is a sketch depth (height) ;

$width_{sketch}$ is a sketch width.

It is worth noting that the width of the sketch limits the error value (4), and the height (depth) controls the probability that the score will break this limit (5) [47], [59].

2.4. Comparison of PDS for further work

Let's compare the analyzed PDS. The comparison is shown in tabular form in Table. 1.

Table 1
PDS comparison

Comparison criteria	Bloom Filter	MinHash	Count-Min Sketch
Required number of hash functions	Low	High	Low
Complexity of hash functions	Low	Low	Low
Sketch size	Medium	Big	Small
Frequency of false positives	Very low	Low	Low
Ability to add / / delete items	Requires modifications	Missing	Maybe

As can be seen from the tabular comparison, all the analyzed PDS have certain disadvantages, which may become certain limitations for their further practical use during neurosynthesis [47], [60]. So for Bloom Filter, there are restrictions on adding or removing elements. This problem is solved in the latest modifications, but such mechanisms significantly complicate the work. MinHash requires a large number of hash functions while running, and while they may be quite simple, calculating them still takes up some time resources. Moreover, when using a large number of such hash functions, the sketch of the resulting view also becomes more complicated. Count-Min Sketch looks like the best option in this comparison, precisely because of the low complexity of the sketch and low requirements for the number and complexity of hash functions. However, it should still be noted that there is a higher probability of false triggering (although it is also noted in other PDS cases) [47], [60].

Therefore, we will plan the further use scheme based on these shortcomings.

3. Materials and methods

Based on the above-mentioned features, subprocesses of neurosynthesis and analysis of the most popular PDS, we will determine the scheme of implementation of PDS in the process of neuroevolutionary synthesis of ANN:

- Bloom Filter will be used when sending information about the ANN between the cores of a parallel system;
- MinHash is used in the sequential execution of neuroevolutionary synthesis to encode genetic information about individuals, to assess acceleration at the breeding stage;
- Count-Min Sketch will also be used to encode genetic information about individuals during neuroevolution synthesis.

Previously, it has been noted that all genetic information about neural networks is encoded using direct coding sequencing [61-64]. In general, this approach is based on coding connections: the genotype of an individual presents information about the weights (WW) of interneuronal connections of the neuromodel, but each gene will contain information about the indices of the initial (N_i) and final (N_o) communication neuron, as well as its weight. In the case when the method works with recurrent neural networks, an additional cell with the feedback weight is added, its index is determined by the index of the original neuron.

When using Bloom filter, four hash functions will be used, which will be used to sequentially encode values from an array of interneuronal relationships [65]:

$$H_{1(N_i)} \oplus H_{2(N_o)} \oplus H_{3(WW)} = f(WW), \tag{6}$$

So, to find the value of $f(WW)$, hash WW four times, perform three table searches, and exclude-or combine-four values. In practice, four v hashes can be reduced to a single 64-bit or 128-bit hash, which is divided into four 16-bit or 32-bit values, respectively.

Such an operation will be performed with the best individuals before sending them from parallel cores to the main thread of the parallel system to investigate the reduction of overhead costs.

MinHash will be used for coding and subsequent comparison in sequential neuroevolution synthesis. However, since adding or removing data from such a structure is not possible (or not rational due to computational costs), the population size will increase and the number of epochs will decrease [57].

When using Count-min Sketch, an array of data about the ANN is represented as a specific sketch and decoded afterwards as at Figure 6.

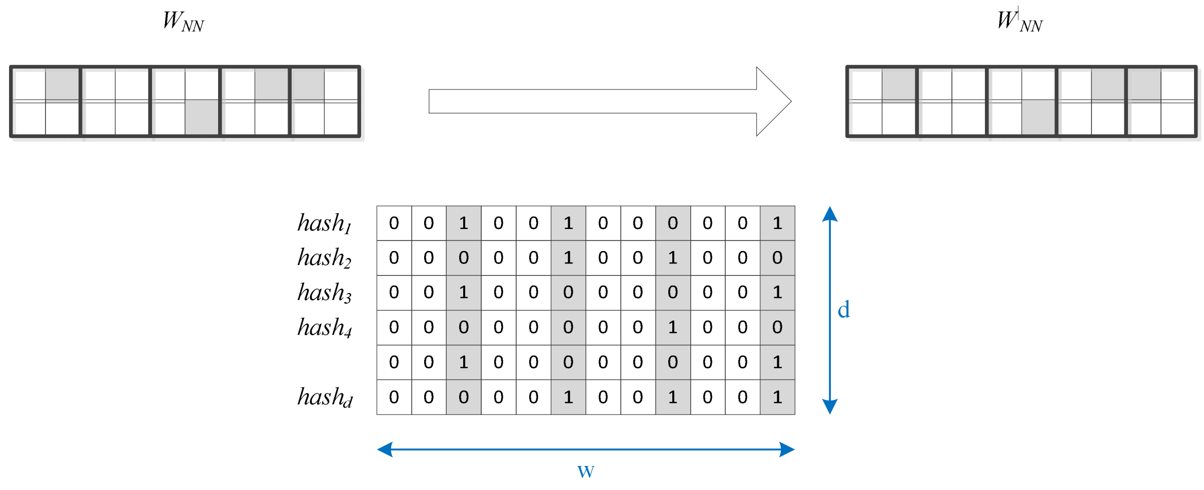

Figure 6: Encoding and decoding information about the ANN using Count-min sketch

Thus, the proposed encoding method begins with an initial representation of a matrix constructed from so called polymer cells containing information about the input-output connections between neurons and the weights of such connections.

Next, hash functions are defined (by default, we recommend taking 4 hash functions) and a matrix is created for their output, as shown in Figure6.

After that, hash outputs are calculated for each element of the ANN data stream and the corresponding counter in the matrix is increased.

Thus, increasing the corresponding counts in the matrix, we get an updated matrix.

In some cases, due to a hash collision, it is possible that the received frequency of the element is slightly higher than expected. The encoding accuracy will depend on how unique the hash functions that return the element value are. Also, the larger the hash function, the more accurate the frequency will be.

In this case, the probabilistic data structure Count–min sketch allows you to calculate the frequency of big data flows in sublinear space with an estimated time complexity, that is $O(1)$, with a constant time component that does not depend on the parameters of the neural network model.

4. Experimental research

For an experimental study of the work, it was decided to choose a moderate sample of data as a sample, which would not initially require additional complications of neural networks, since this can become a problem in the case of using MinHash. test it on two tasks that would differ in scale. This is how the Tic-Tac-Toe Endgame Data Set was selected [66-69]. The main characteristics of the sample are shown in Table 2.

As a parallel computing system, the equipment of Department of the software tools the National university "Zaporizhzhia polytechnic" was used: Xeon processor E5-2660 v4 (14 cores), RAM 4x16 GB DDR4, the programming model of Java threads.

Table 2
Characteristics of the tic-Tac-Toe endgame data set sample

Tic-Tac-Toe Endgame Data Set			
Data Set Characteristics:	Multivariate	Number of Instances:	958
Attribute Characteristics:	Categorical	Number of Attributes:	9

To test the MinHash case, the metaparameters specified in Table 3 will be set.

Table 3
Metaparameters for the MinHash use case

Comparison criteria	Bloom Filter
Population size	100
Number of training epochs	1
Elite size	5%
Activation function (fitness functions)	hyperbolic tangent
Probability of mutation	25%
Crossover type	two-point
Types of mutation	deleting an interneuronal connection
	removing a neuron
	adding interneuronal connection
	adding a neuron
	changing the activation function

For testing the Count–min sketch case, the main parameters remain unchanged, but the number of epochs increases to 5.

When testing the Bloom Filter use case, not only the meta parameters of the method were met, but also the hardware usage parameters specified in Table 4.

Table 4
Metaparameters for the Bloom Filter use case

Comparison criteria	Bloom Filter
Population size	100
Number of training epochs	5
Elite size	5%
Activation function (fitness functions)	hyperbolic tangent
Probability of mutation	25%
Type of crossover on parallel threads	two-point
Type of crossover on the main thread	uniform (two-point)
Types of mutation	deleting an interneuronal connection
	removing a neuron
	adding interneuronal connection
	adding a neuron
	changing the activation function
Number of cores (threads)	14

The test results for the tic-tac-Toe Endgame data set sample presented using MinHash are shown in Table 5.

The test results for the tic-tac-Toe Endgame data set sample presented using Count-Min Sketch are shown in Table 6.

Table 5
Test results using MinHash

Method	Synthesis Time, s	Time at the selection stage, s	Memory usage (at the selection stage)	Error in the training sample	Error in the test sample
MGA	2487	561	91%	0.11	0.24
MGA + MinHash	2189	287	61%	0.11	0.25

Table 6
Test results using Count-Min Sketch

Method	Synthesis Time, s	Time at the selection stage, s	Memory usage (at the selection stage)	Error in the training sample	Error in the test sample
MGA	5364	1742	94%	0.024	0.13
MGA + Count-Min Sketch	4865	1025	56%	0.029	0.15

The test results for the tic-tac-Toe Endgame data set sample presented using Bloom filter are shown in Table 7.

Table 7
Test results using Bloom filter

Method	Synthesis Time, s	Communication overhead	Average time of communication overhead	Error in the training sample	Error in the test sample
MGA	2494	0.4	199.5	0.022	0.1
MGA + Bloom filter	1867	0.28	104.6	0.03	0.174

5. Analysis of experimental results

Analyzing the presented results, we can come to the following conclusions. When using MinHash, there were no significant improvements: it was at the selection stage that memory usage decreased from 92% to 61%, but up to this point, memory usage has not changed. Also, the acceleration was only at the selection stage, namely from 1742 s to 1025 s. However, the time for encoding and decoding networks increased significantly, precisely because of the complexity of this process, so the optimization was rather point-based and cannot be recommended for use in the future. In addition, we should mention the low accuracy due to the reduction in the number of epochs and the inability to use crossbreeding.

When using Count-min Sketch, the acceleration situation was similar, because during the execution of the method, the acceleration was noticeable, but the process of encoding and decoding individuals took too long, so the total synthesis time is almost the same. But memory usage has really become justifiably optimized from 94% to 56%. This can be explained by the fact that, unlike MinHash, almost during the entire process, in addition, such encoding made it possible to perform mutation and crossing without significant problems. On the other hand, we should note a slight decrease in the accuracy of operation (due to false positives of the decoded network).

Using Bloom filter made it possible to reduce the share of shipments and their time, because data was sent from cores in a compact format. In addition, decoding networks on the main core did not impose additional complex calculations on parallel cores. However, even this strategy did not significantly improve the speed of work. Moreover, although uniform crossing was chosen on the main core, during the development of information modules, the problem of organizing multi-parent crossing arose. Therefore, unfortunately, it was necessary to use uniform crossover as a two-point crossover.

In general, it can be seen that the most promising is the use of Count-min Sketch, because this made it possible to perform the full synthesis process without additional complications.

6. Conclusions

The paper examines the variants and possibilities of using PDS in neuroevolutionary synthesis of ANNs, which can significantly reduce memory usage during synthesis and speed up this process.

The scientific novelty lies in the fact that during neuroevolutionary synthesis at different stages and with different execution schemes (sequential and parallel), it is proposed to use different PDS. This made it possible to study options for reducing resource intensity pointwise and in general. Studies have shown that despite the popularity of PDS for preserving big data in neuroevolutionary synthesis, this approach meets a number of obstacles. Moreover, during the work, recommendations were developed for the use of individual PDS for individual stages and approaches.

The practical significance lies in the fact that practical problems of coding the ANN are solved, which can later be used for diagnostics, forecasting, evaluation and modeling. The results of the experiments showed that the proposed coding methods make it possible to encode information about the ANN more compactly for its subsequent transmission to workstations for use as a model for diagnosis, prediction, evaluation and modeling.

Prospects for further research and development areas include a detailed analysis of the use of the Count-Min sketch data structure. This is due to the most acceptable results when testing various PDS and approaches to their use.

7. Acknowledgements

The work was carried out with the support of the state budget research projects of the state budget of the National University “Zaporozhzhia Polytechnic” “Intelligent methods and software for diagnostics and non-destructive quality control of military and civilian applications” (state registration number 0119U100360) and “Development of methods and tools for analysis and prediction of dynamic behavior of nonlinear objects” (state registration number 0121U107499).

8. References

[1] M. Elgendy, Deep Learning for Vision Systems, New York, Manning Publications, 2020.
[2] D. Hudgeon, R. Nichol, Machine Learning for Business: Using Amazon SageMaker and Jupyter, Manning Publications, 2020.
[3] D. Barh, Artificial Intelligence in Precision Health: From Concept to Applications, Academic Press, Cambridge, 2020.
[4] K. Warr, Strengthening Deep Neural Networks: Making AI Less Susceptible to Adversarial Trickery, O'Reilly Media, Massachusetts, 2019.
[5] Y. Goldberg , G. Hirst, Neural Network Methods in Natural Language Processing (Synthesis Lectures on Human Language Technologies), Morgan and Claypool Publishers, California, 2017.
[6] T. Neskorodieva, E. Fedorov, Method for Automatic Analysis of Compliance of Expenses Data and the Enterprise Income by Neural Network Model of Forecast, in: Proceedings of the

Mathematical Modeling and Simulation of Systems (MODS'2020), Springer (2020), 156-165. doi: 10.1007/978-3-030-58124-4_15

[7] J.A.J. Alsayaydeh, W.A.Y. Khang, A.K.M.Z Hossain, V. Shkarupylo, J. Pusppanathan, The experimental studies of the automatic control methods of magnetic separators performance by magnetic product, ARPN Journal of Engineering and Applied Sciences, vol. 15(7) (2020) 922-927.

[8] J.A.J. Alsayaydeh, W.A. Indra, W.A.Y. Khang, V. Shkarupylo, D.A.P.P. Jkatisan, Development of vehicle ignition using fingerprint, ARPN Journal of Engineering and Applied Sciences, vol. 14(23) (2019) 4045-4053.

[9] J.A.J. Alsayaydeh, W.A.Y. Khang, W.A. Indra, J.B. Pusppanathan, V. Shkarupylo, A.K.M. Zakir Hossain, S. Saravanan, Development of vehicle door security using smart tag and fingerprint system, ARPN Journal of Engineering and Applied Sciences, vol. 9(1) (2019) 3108-3114.

[10] J.A.J. Alsayaydeh, W.A.Y. Khang, W.A. Indra, V. Shkarupylo, J. Jayasundar, Development of smart dustbin by using apps, ARPN Journal of Engineering and Applied Sciences, vol. 14(21) (2019) 3703-3711.

[11] J.A. Alsayaydeh, M. Nj, S.N. Syed, A.W. Yoon, W.A. Indra, V. Shkarupylo, C. Pellipus, Homes appliances control using bluetooth, ARPN Journal of Engineering and Applied Sciences, vol. 14(19) (2019) 3344-3357.

[12] Google, 2021. URL: https://www.google.com/

[13] S. Dresden, Search engine with neural network weighting based on parametric user data, 2004. Patent No. US20050004905A1, Filed March 3rd., 2003, Issued June 6th., 2005.

[14] Search Engines and Neural Networks, 2018. URL: https://towardsdatascience.com/https-towardsdatascience-com-search-engines-and-neural-networks-97e0df4f088d

[15] W. Serrano, Neural Networks in Big Data and Web Search. Neural networks in web search, 4(1):7 (2018) 1-41. doi: 10.3390/data4010007

[16] Bing, 2021. URL: https://www.bing.com/

[17] Microsoft to accelerate Bing search with neural network, 2015. URL: https://www.extremetech.com/extreme/199814-microsoft-to-accelerate-bing-search-with-neural-network

[18] Baidu, 2021. URL: http://www.baidu.com/

[19] YATI & ERNIE: Machine Learning in Yandex and Baidu, 2021. URL: https://www.searchenginejournal.com/yati-ernie-machine-learning-yandex-baidu/392982/

[20] V.Mnih, A.P. Badia, M. Mirza, A. Graves, et al., Asynchronous methods for deep reinforcement learning, in: Proceedings of the 33rd International Conference on International Conference on Machine Learning, ICML'16, 48 (2016) 1928–1937.

[21] L. Tai, J. Zhang, M. Liu, J. Boedecker, W. Burgard, A Survey of Deep Network Solutions for Learning Control in Robotics: From Reinforcement to Imitation, 2018. URL: https://arxiv.org/abs/1612.07139

[22] R. Gilman, K. Wang Intuitive RL: Intro to Advantage-Actor-Critic (A2C), 2018. URL: https://hackernoon.com/intuitive-rl-intro-to-advantage-actor-critic-a2c-4ff545978752

[23] J. K.H. Franke, G. Köhler, N. Awad, F. Hutter Neural Architecture Evolution in Deep Reinforcement Learning for Continuous Control NeurIPS 2019 MetaLearn Workshop, 2019. URL: https://arxiv.org/abs/1910.12824

[24] Reinforcement Learning: Deep Q Networks, 2020. URL: https://blogs.oracle.com/datascience/reinforcement-learning-deep-q-networks

[25] A. Juliani Simple Reinforcement Learning with Tensorflow Part 8: Asynchronous Actor-Critic Agents (A3C), 2016. URL: https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2

[26] P. Dangeti, Statistics for Machine Learning: Techniques for exploring supervised, unsupervised, and reinforcement learning models with Python and R, Packt Publishing, Birmingham, 2017.

[27] B. Bateman, A. R. Jha, B. Johnston, I. Mathur, The Supervised Learning Workshop: A New, Interactive Approach to Understanding Supervised Learning Algorithms, 2nd Edition, Packt Publishing, Birmingham, 2020.

[28] X. Zhu, A.B. Goldberg, Introduction to Semi-Supervised Learning (Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan and Claypool Publishers, California, 2009.

[29] G. Hinton, Unsupervised Learning: Foundations of Neural Computation (Computational Neuroscience), Bradford Books, Massachusetts, 1999.
[30] A. Jones, C. Kruger, B. Johnston The Unsupervised Learning Workshop: Get started with unsupervised learning algorithms and simplify your unorganized data to help make future predictions, Packt Publishing, Birmingham, 2020.
[31] R.S. Sutton, A.G. Barto, Reinforcement Learning, second edition: An Introduction (Adaptive Computation and Machine Learning series) second edition, Bradford Books, Massachusetts, 2018.
[32] P. Winder, Reinforcement Learning: Industrial Applications of Intelligent Agents, O'Reilly Media, Massachusetts, 2020.
[33] L. Graesser, W.L. Keng, Foundations of Deep Reinforcement Learning: Theory and Practice in Python (Addison-Wesley Data & Analytics Series), Addison-Wesley Professional, Boston, 2019.
[34] K. O. Stanley, J. Clune, J. Lehman, et al., Designing neural networks through neuroevolution. Nature Machine Intelligence 1 (2019) 24–35. doi: 10.1038/s42256-018-0006-z
[35] Omelianenko, Hands-On Neuroevolution with Python: Build high-performing artificial neural network architectures using neuroevolution-based algorithms, Packt Publishing, Birmingham, 2019.
[36] A. Bergel, Agile Artificial Intelligence in Pharo: Implementing Neural Networks, Genetic Algorithms, and Neuroevolution, Apress, New York, 2020.
[37] A. Gaier, David Ha Exploring Weight Agnostic Neural Networks, 2019. URL: https://ai.googleblog.com/2019/08/exploring-weight-agnostic-neural.html
[38] K O. Stanley, J. Clune Welcoming the Era of Deep Neuroevolution, 2017. URL: https://eng.uber.com/deep-neuroevolution/
[39] A.O. Oliinyk, T.A. Zaiko, S.A. Subbotin, Factor analysis of transaction data bases. Automatic Control and Computer Sciences 48(2) (2014) 87-96. doi: 10.3103/S0146411614020060
[40] A. Oliinyk, S. Skrupsky, S.A. Subbotin, Parallel computer system resource planning for synthesis of neuro-fuzzy networks. Advances in Intelligent Systems and Computing, 543 (2017) 88-96. doi: 10.1007/978-3-319-48923-0_12
[41] A. Oliinyk, S. Skrupsky, S. Subbotin, I. Korobiichuk, Parallel method of production rules extraction based on computational intelligence. Automatic Control and Computer Sciences, 51(4) (2017) 215-223. doi: 10.3103/S0146411617040058
[42] A. Oliinyk, S. Skrupsky, S. Subbotin, Experimental research and analysis of complexity of parallel method for production rules extraction. Automatic Control and Computer Sciences, 52 (2) (2018) 89-99. doi: 10.3103/S0146411618020062
[43] S. Leoshchenko, A. Oliinyk, S. Subbotin, T. Zaiko, N. Gorobii, Implementation of selective pressure mechanism to optimize memory consumption in the synthesis of neuromodels for medical diagnostics, in: Proceedings of the 2nd International Workshop on Informatics & Data-Driven Medicine, IDDM 2019, CEUR-WS, 2019, pp. 109-120. DBLP Key: conf/iddm/LeoshchenkoOSZG19
[44] A. Gakhov, Probabilistic Data Structures and Algorithms for Big Data Applications, Books on Demand, 2019.
[45] H. Knebl, Algorithms and Data Structures: Foundations and Probabilistic Methods for Design and Analysis, Springer, Berlin, 2020.
[46] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, The MIT Press, Cambridge, 2009.
[47] Probabilistic data structures for web analytics and data mining, 2012. URL: https://highlyscalable.wordpress.com/2012/05/01/probabilistic-structures-web-analytics-data-mining/
[48] Hydra, 2014. URL: https://github.com/addthis/hydra
[49] Big Data Hadoop Alternatives: What They Offer and Who Uses Them, 2018. URL: https://datafloq.com/read/Big-Data-Hadoop-Alternatives/1135
[50] Apache Hadoop, 2021. URL: https://hadoop.apache.org/
[51] NetApp, 2021. URL: https://www.netapp.com/
[52] Big Data Analytics solutions: The bigger the better, 2021. URL: https://www.netapp.com/artificial-intelligence/big-data-analytics/

[53] B.H. Bloom, Space/time trade-offs in hash coding with allowable errors. Communications of the ACM T. 13(7) (1970) 422–426. https://doi.org/10.1145/362686.362692
[54] Bloom Filters: Is element x in set S?, 2017. URL: https://www.abhishek-tiwari.com/bloom-filters-is-element-x-in-set-s/
[55] How to count Big Data: Probabilistic data structures and algorithms, 2019. URL: https://www.kdnuggets.com/2019/08/count-big-data-probabilistic-data-structures-algorithms.html
[56] Introduction to Probabilistic Data Structures, 2015. URL: https://dzone.com/articles/introduction-probabilistic-0
[57] A.Z. Broder, On the resemblance and containment of documents. Compression and Complexity of Sequences: Proceedings, IEEE (1997) 21–29. doi:10.1109/SEQUEN.1997.666900
[58] A.Z. Broder, M. Charikar, A.M. Frieze, M. Mitzenmacher, Min-wise independent permutations, in: Proceedings of the 30th ACM Symposium on Theory of Computing (STOC '98), New York, NY, USA: Association for Computing Machinery (1998) 327–336. doi:10.1145/276698.276781
[59] G. Cormode, Count-Min Sketch, Encyclopedia of Database Systems, Springer, 2009, 511-516.
[60] I. Katsov, Introduction to Algorithmic Marketing: Artificial Intelligence for Marketing Operations, Books on Demand, 2017.
[61] S. Leoshchenko, A. Oliinyk, S. Subbotin, T. Zaiko, S. Shylo, V. Lytvyn, Sequencing for encoding in neuroevolutionary synthesis of neural network models for medical diagnosis, in: Proceedings of the 3rd International Conference on Informatics & Data-Driven Medicine, IDDM 2020, CEUR-WS, 2020, pp. 62-71.
[62] B.A. Pierce, Genetics: A Conceptual Approach, W.H. Freeman & Co. Ltd., New York, 2019.
[63] R. Brooker, Genetics: Analysis and Principles, McGraw-Hill Education, New York, 2020.
[64] S. Leoshchenko, A. Oliinyk, S. Subbotin, N. Gorobii, T. Zaiko, Synthesis of artificial neural networks using a modified genetic algorithm, in: Proceedings of the 1st International Workshop on Informatics & Data-Driven Medicine, IDDM 2018, CEUR-WS, 2018, pp. 1-13. DBLP Key: conf/iddm/PerovaBSKR18
[65] B. Reagen, U. Gupta, R. Adolf, M. Mitzenmacher, A. Rush, G.-Y. Wei, D. Brooks, Weightless: Lossy Weight Encoding For Deep Neural Network Compression, Computer Science, Mathematics, 2017. URL: http://proceedings.mlr.press/v80/reagan18a/reagan18a.pdf
[66] Tic-Tac-Toe Endgame Data Set, 1991. URL: https://archive.ics.uci.edu/ml/datasets/Tic-Tac-Toe+Endgame
[67] C. J. Matheus, ,L. A. Rendell Constructive induction on decision trees. Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Detroit, MI: Morgan Kaufmann, 1989, pp. 645-650.
[68] C. J. Matheus Adding domain knowledge to SBL through feature construction. Proceedings of the Eighth National Conference on Artificial Intelligence,Boston, MA: AAAI Press, 1990, pp. 803-808.
[69] D. W. Aha Incremental constructive induction: An instance-based approach. In Proceedings of the Eighth International Workshop on Machine Learning, Evanston, ILL: Morgan Kaufmann, 1991, pp. 117-121.