Neural Network Software Technology Trainable on the Random Search Principles

Victor Krasnoproshin and Vadim Matskevich
Belarusian State University
Belarus, Minsk, Nezavisimosti av. 4, 220030
Email: krasnoproshin@bsu.by, matskevich1997@gmail.com

***Abstract*—The paper deals with a state-of-art neural technology programmed implementation problem in which the training process is based on random search algorithms.**

Training neural networks is a typical optimization problem. At the initial stage of neural network technologies development, various variants of gradient methods were traditionally used to solve such problems. Such methods, as a rule, met the requirements for the problem in terms of quality and speed of training. However, with the appearing of a new class of applied problems, the situation has changed. The traditional approach to training using gradient methods did not always meet the requirements of the applied problem in terms of the resulting solution quality.

The paper proposes one of the options for the software implementation of neural network technology (in the form of a framework) according to the ostis 2021 standard, in which random search algorithms are used to train neural networks.

***Keywords*—framework, neural network, training, random search algorithms, annealing method**

I. Introduction

In modern society, digital data processing technologies based on artificial intelligence methods are rapidly developing. In particular, neural network technologies based on various artificial neural networks architectures have become widespread.

Due to their high flexibility and the ability to tune to the subject area, they are actively used to solve a wide class of applied problems. However, neural network tuning for the problem being solved (training) is a time-consuming process.

Automation of the training process is effectively solved within the existing frameworks [1]. They allow us to simplify the neural networks training process by using already implemented training algorithms. The use of gradient optimizers inside such frameworks is quite justified. Gradient methods have a high convergence rate and in practice provide an acceptable solution quality obtained. When developing the first automated neural networks training systems, there wasn't a wide variety of computing devices, so they didn't have cross-platform property. However, with the computing technology development, more and more calculations are transferred from the central processor to connected computing devices. This allows us to use simultaneously a large number of devices and significantly increase the efficiency of computing. Moreover, modern frameworks have become cross-platform. However, as digital technologies develop, the class of applied problems for which the solution quality obtained is critical is constantly expanding.

It should be noted here that many modern frameworks use gradient optimization methods, which do not always guarantee the optimal solution achievement. Consequently, when solving such applied problems, they are not effective enough, which makes the problem of developing a software package with alternative training methods up to date.

The paper proposes a framework's software implementation variant, in which random search algorithms are used to train neural networks.

II. Problem analysis

Currently, a wide range of applied problems is solved using neural network technologies implemented in the form of frameworks. This technology is a set of software and algorithmic tools that implement the architectures of various types of neural networks focused on solving various classes of applied problems.

Today, there are a number of frameworks for solving machine learning problems. Among the most popular, in particular, the following can be distinguished.

MXNet is a high-performance and cross-platform framework that is widely used in solving applied problems. However, this framework has certain drawbacks. This is not a very convenient user interface compared to simpler frameworks and a rather meager range of optimization algorithms. It only supports some modifications of gradient descent. This framework completely lacks support for random search methods.

Tensorflow 2. is cross-platform and has a simple user interface. Currently, it is the most common framework for applied problems solving. Supports learning with various gradient methods and genetic algorithm. The disadvantages include insufficiently high performance, since it contains the costs of high-level programming languages and a poor variety of non-directional optimization algorithms.

Caffe 2 is a high performance cross platform framework. However, it lacks support for recurrent neural networks and non-directional training algorithms. This framework significantly limits the class of constructed neural networks and has all the disadvantages of gradient methods.

Thus, today the following disadvantages are typical for modern frameworks.

Most of them either have a limited set of training algorithms, or support a limited neural networks class, or have low performance.

III. Framework architecture and structure

Consider a variant of a software package implemented in a form of framework in which random search algorithms are used to train neural networks.

The software package was developed in C++ using the OpenMP and OpenCL libraries. OpenMP libraries provide efficient organization of parallel computing within a single processor, while OpenCL provides compatibility and parallel computing on a wide class of computing devices.

According to the ostis 2021 standard, the framework can be described as follows:

framework
$\Rightarrow$ *decomposition**:
{
- *algorithms library*
- *train behaviour parameters*
- *architecture library*
- *database*
- *compress/decompress module*
- *predict module*
- *load/save module*

}

algorithms library
$\ni$ *FAST_ANNEALING*
$\ni$ *SLOW_ANNEALING*
$\ni$ *GENETIC*
$\ni$ *SGD*
$\ni$ *MOMGRAD*
$\ni$ *ADAM*
$\ni$ *FTML*

train behaviour parameters
$\ni$ *NO_TRAIN*
$\ni$ *JUST_TRAIN*
$\ni$ *NEW_TRAIN*
$\ni$ *CONTINUE_TRAIN*

architecture library
$\ni$ *RBM_BERNOULLI_BERNOULLI*
$\ni$ *RBM_GAUSS_BERNOULLI*
$\ni$ *AUTOENCODER*
$\ni$ *PERCEPTRON_NN*
$\ni$ *CONV_LAYER*
$\ni$ *POOLING_LAYER*

PERCEPTRON_NN
$\Rightarrow$ *part**:
ActivationFunction

AUTOENCODER
$\Rightarrow$ *part**:
ActivationFunction

ActivationFunction
$\ni$ *NONE*
$\ni$ *BIPOLYARSIGM*
$\ni$ *SIGM*
$\ni$ *ReLU*
$\ni$ *SOFTMAX*

The developed software package consists of the following main modules: two libraries (algorithms and architectures of neural networks), a database and a database with configuration files (for setting up algorithms from the library), modules (for execution on connected computing devices and the neural networks functioning), and finally, user interaction interface.

The database contains all the necessary data sets for training neural networks. The data is loaded at the user request. The framework has built-in methods for generating various types of samples (from the requested data) for training and testing neural networks.

The algorithm library contains a wide range of different optimization algorithms: simple gradient, moment and adaptive moment methods, following the moving leader method, genetic algorithm and annealing method. Configuration files are loaded during the framework initialization and contain sets with optimal parameter values, which, if necessary, are recalculated taking into account the neural network architecture. Execution Modules on connected computing devices are loaded as they are found. The framework also has a high degree of flexibility, it can be executed on a limited number of processor threads (the limit can be adjusted), on several connected computing devices. In addition, a completely single-threaded execution mode is possible.

The framework allows assembly without the use of parallel computing on connected devices and the processor. For this, the constants.h file contains the DISABLE_OPEN_CL, DISABLE_OPENMP constants. When set to non-zero values, the framework disables the ability to use parallel computing. This allows the framework to be independent of the settings and computer configuration. The framework has additional constants ENABLE_FAST_MATH and ENABLE_NORMAL_DISTR. They allow us to activate the possibility of using accelerated trigonometric functions, for example, to generate a normal distribution. They also activate the tabled generator of basic random variables. The size of the table in the framework is more than a billion, so the random numbers repetition is not critical. To speed up trigonometric functions, the segment of valid argument values is divided into a fixed number of parts (the accuracy is controlled by parameters). For each part, the value of cosine and sine is calculated and stored in a special table. The acceleration of calculations is achieved in the many times repeated use of tabular approximate values instead of the full calculation of the function value.

To train all non-recurrent networks with gradient methods, the back-propagation algorithm is used to calculate the gradient. For restricted Boltzmann machines, the gradient calculation uses the Constructive Divergence (CD-1) algorithm. Then, the user-selected gradient modifications are applied to the result.

The user interface contains a fairly wide range of functionality, covering all the procedures necessary for convenient framework use. This is loading (saving) a neural network from a hard disk, constructing neural networks based on the architecture base, forming a training dataset based on selected objects classes and data. Neural networks training process implementation, using various types of algorithms. Solving applied problems, color image compression and object recognition.

The neural networks training process can be described as follows. First, the user loads the training data from the database and calls the building training dataset methods (if needed). It then declares the neural network skeleton on which the neural network will be built. To do this, the framework implements a special fakeDeepNN data type. Using the prebuildDeepNN method, the user tells the designed neural network the images resolution, the images type (black and white, grayscale or color), indicates whether the network will be convolutional and assigns (if necessary) the future network a serial number.

Then, a neural network is constructed to the created skeletone (by sequentially adding different types of layers). The need for layer decomposition is indicated when adding the first layer of the network. In other layers, the decomposition size is calculated automatically based on the layer sizes entered by the user.

At the end of the neural network design process, the user declares an object of deepNN type and calls the buildDeepNN method. Passes the previously created network skeleton as a parameter. This function, in addition to connecting individual fragments into a single network, checks the correctness of the constructed network. For example, the incompatibility network layers' sizes, the incompatibility of layers' types (for example, one layer generates continuous data, and the next layer receives discrete data, or vice versa). If an error is found, the program generates the appropriate error and exits.

In the next step, the user sets the neural network training settings. The framework is highly flexible and allows for many different training modes. To do this, it declares an object of a special type TrainingSettings and, by calling the object's method for adding settings for a separate layer, sequentially sets the training settings for all layers of the network. In the training settings of a separate layer, the following options are set: optimization algorithm, training style, layer training time, number of objects in the training and validation sets.

Setting the training style allows you not to train individual neural networks' layers, but to load network's trained fragments from the hard disk. The framework also supports the partial training option. To do this, the training style indicates: simple training or initial training is carried out — after which additional training of the layer is allowed, or continuation of training — the layer continues training. This option is useful when training large neural networks or when using low-power computing devices.

In the case of initial training, in addition to the trained network, the training state of a separate layer is stored. The description of the learning state depends on the optimizer chosen. In the case of continuing training, in addition to the initial training, the saved learning state of the neural network layer and the partially trained layer are loaded.

The training time is set separately for each layer. Inside all optimizers, there is a built-in implementation of the timer function that controls the time spent on training the layer. As soon as the time allocated for training has expired, the training process for this layer is completed.

The sizes of the training and validation sets allow us to control the amount of data required for layer-by-layer neural network training.

After creating a neural network and setting up the training of its individual layers, the user calls the constructed network training function using the trainDeepNN function. With this function, the user informs about the network being trained, training settings, some information about the input data and the need to use external devices for training.

The trainDeepNN function is key to the framework. At the beginning of the execution, if necessary, it scans the connected computing devices and initializes the executable modules on them. The function then checks that the the network layers training settings are correct. For example, it checks for the trained layers presence, that do not require training or will be retrained, etc. This function loads all settings of the entire algorithms library and optimizer selected by the user. Inside this function, a complex interaction with the input data for their decomposition (if necessary), linking the layers of the trained network, creating and deleting service buffers for the optimizer calling the training of the corresponding layer type by the corresponding optimizer, and other technical details are implemented. This method, at the end of training, saves the trained network to the hard disk.

The user can also call the compressImages and predict functions. Both functions take a trained neural network and input data. The first of them performs data compression and returns their compressed image, the second returns a set of labels. The number of labels corresponds to the number of sended images. The label specifies the object class that the neural network has detected in the image.

IV. Architectures library

The architectures library fully fits into the ostis system described in [2] [3]. The software package supports the following types of neural networks: restricted Boltzmann machine of Gauss-Bernoulli and Bernoulli-Bernoulli types, autoencoders, decomposition of network layers from the above types, multilayer perceptrons, convolution layers, pooling layers.

A small number of frameworks support restricted Boltzmann machines, although they are used for key frames detecting in video sequences [4] filtering data [5], encoding key phrases in information retrieval [6] [informational retrieval]. Based on supported autoencoders and restricted Boltzmann machines, the framework allows us to design deep belief networks for data compression and preprocessing for subsequent data classification.

Support for sampling and convolution layers, multilayer perceptrons allows us to design deep convolutional neural networks to build neural network classifiers that are used for a wide class of applied problems.

Thanks to the support for the decomposition of individual layers, the framework allows you to significantly reduce the number of tunable network parameters, reduce the required amount of data for training, and increase the efficiency of parallelizing the neural network layers training.

The supported wide library of architectures allows us to design neural networks of almost any architecture, which makes the framework a universal tool for neural network data processing.

V. Training algorithm based on the annealing method

To solve the problem of efficiency of neural networks training algorithms, there are algorithms based on random search. The annealing method is the most promising random search algorithm for training neural networks. The paper proposes the following training algorithm based on the annealing method.

Let an objective function *F* be defined on a finite set of admissible solutions Ω and for each element $x \subset \Omega$ of which the set of neighboring elements is $N(x) \subset \Omega$ given. The conditional optimization problem in this case can be specified as a triple (Ω, *F*, *N*). Let us consider the possibilities of its solution using the annealing method. The algorithm includes the following main steps.

Preliminary stage. Initialization of the neural network initial state $Net_0 = Net(x_{10}, x_{20}, \ldots, x_{m0})$ and temperature sequences $T_0, T_1, \ldots, T_k$, related by the ratio:

$$T_k = \frac{T_0}{ln(k+2)}, k > 0,$$

where T_0 - present value.

General k-th iteration.

Step 1. Random value generation. Generated *m* uniformly distributed on the segment from zero to the number of parameters in the set of discrete random variables $a_1, a_2, \ldots, a_m$. Generated *m* random permutations of length equal to the number in the set of parameters. The first $a_1, a_2, \ldots, a_m$ elements of each permutation define the indexes of the parameters to be changed in each set of parameters, respectively.

Step 2. New solution generation. For each changing parameter, a uniformly distributed on the segment [-*l*/2;*l*/2] random value *b* is generated. Value *l* depends on what set is changing parameter belongs to and equal to $l_1, l_2, \ldots, l_m$ respectively. Values *l* for each set are given as algorithm parameters.

Let x_i is changing parameter and $x_i^{'}$ is its new value, then:

$$x_i^{'} = x_i + b$$

Step 3. Transition principle. Let *x* be a current solution and *y* was generated on step 2 as new solution. Then solution $x^{'}$ on the next iteration is determined in a following way:

$$P(x^{'} = y|x) = \min\{1, exp(\frac{F(x) - F(y)}{T_k})\}$$

Step 4. Stop criteria. If the time for training the neural network has expired, then the algorithm ends. Otherwise, the transition to the next iteration is performed.

Previously, it was proved that the proposed algorithm converges in probability to the optimal solution, and from any initial solution [7].

VI. Genetic algorithm for neural networks training

Additionally, a special genetic algorithm modification for neural networks training was developed for the framework.

Preliminary stage. Generation of several random solutions. Each individual solution is a complete neural network, the architecture of which is set before training algorithm running and doesn't change. The number of solutions N is a parameter of the algorithm. For each solution, the value of the quality functional to be optimized is calculated.

General k-th iteration.

Step 1. The worst one is chosen from the current set of solutions.

Step 2. For the chosen solution, a "mutation" is performed – generation of a new solution from the current one according to the same scheme as for the developed annealing algorithm. The only difference is that the values of the annealing parameters may differ from the genetic algorithm.

Step 3. For the obtained solution, the quality functional value is calculated.

Step 4. If the value of the obtained functional for the new solution is better than for the current one, then the current solution is replaced with a new one, otherwise the new solution is discarded.

Step 5. The solution *b* is randomly selected from the set of current solutions. The best solution *a* is also selected from the set.

Step 6. "Crossbreeding". For all values of the solution parameters *a*, *b*, the following calculations are made.

$$\begin{cases} d_i = b_i - a_i \\ c_i = a_i + d_i * \alpha \\ \alpha \in [0; \phi], 0 < \phi \leq 1 \end{cases}$$

where α is uniformly distributed random variable on the segment, ϕ is an algorithm parameter.

Step 7. For the obtained solution, the quality functional value is calculated.

Step 8. In the set of solutions, the worst one is chosen. If it is worse than the new solution, then it is replaced by a new solution, otherwise the new solution is discarded.

Step 9. Stop criteria check. If the time for training the neural network has expired, then the algorithm ends. Otherwise, the transition to the next iteration is performed.

VII. Software package implementation features

The base of algorithms is developed taking into account the cross-platform framework property. When developing optimizers, the computing devices architectural features were taken into account. For example, video cards are characterized by a large number of low-power computing cores, which requires good scalability from the parallelization algorithm. In addition, the performance of the video card mainly depends on the efficiency of working with video memory, since it is very slow. This imposes requirements on the locality of the data used for the most efficient video card's cache use. It was also necessary to implement the interaction of the video card with the processor for the results collection and other data transfer.

The above requirements have resulted in each optimizer being implemented twice using different algorithms. One set of algorithms is used to train neural networks on a processor, the other set is used on video cards.

The most time-consuming part of training a neural network is moving data through the training layer. The input data is presented in matrix form, as are the parameters of the neural network. In a simplified form, moving data through a network layer is a matrix multiplication of data by network parameters. Thus, to ensure efficient training, it is necessary to implement efficient matrix multiplication.

To ensure the best data caching, a block data multiplication algorithm was used. It is worth noting that improving caching increases the training efficiency both on the video card and on the processor. The block size for the matrices was selected empirically. For the processor, the optimal block size was 25, for the video card — 16. It should be noted that the data size, the amount of data, and the network size are often not a multiple of these values, so incomplete blocks were filled with zeros at the end of the real data. This approach makes it possible to increase the training efficiency by more than 70% on the processor and more than twice on the video card. This fact makes it possible to compensate for the significantly complicated implementation of optimizers and the entire complex as a whole due to the formation and disbanding of data blocks and work inside data matrix blocks.

It is known that random search methods have slow convergence and require a large amount of computation at a single iteration. Therefore, in order to increase the efficiency of annealing training, a calculation hiding approach (when using connected devices for training) was used.

Each iteration of the annealing method, as mentioned above, consists of 4 stages. All stages, except for calculating the functional value for a new solution, are performed by the processor. The most time-consuming stage is the calculation of the functional value and, with a small network architecture, the new solution generation. The computational power of a video card is on average 20 times higher than that of a processor. This leads to the fact that when training small neural networks, a significant part of the time is spent on generating new solutions. All stages in a separate iteration are performed strictly sequentially.

The generation of a new solution consists of two stages: determination of the number and selection of changeable network parameters; changing the values of the selected network parameters.

To reduce the execution time of a single iteration, you can use the fact that the definition and choice of network parameters to be changed don't depend on the stage of making a new decision. This can be explained by the fact that the generated increment to the values of the changing parameters will not change in this case. Changing the solution changes only the initial values of the parameters being changed. In this case, you can use the well-known calculations hiding trick.

To achieve the most efficient use of computing devices, 3 special procedures were designed.

Procedure A1. You can determine the number and select the parameters to be changed and their increments in parallel on the processor, when the video card calculates the value of the quality functional. Under such conditions, already calculated parameter increments are used to generate a new solution already at the next iteration, which reduces the number of sequentially performed calculations.

Procedure A2. For the subsequent optimization of the training algorithm, let us consider in more detail the stage of calculating the functional value.

Let current solution be equal to *x*, and new solution equal to *y*. During the calculation of the functional value on the video card for the solution *y*, the processor simultaneously generates two new solutions $x_1 \in N(x)$ and $y_1 \in N(y)$ where N — set of possible generated solutions from the current. After calculating the value of the functional, the necessity of transition to a new solution is checked. If a new decision is accepted, the next solution to be tested would be y_1 otherwise x_1. This procedure allows you to mask the stage of generating a new solution. Thus, when moving to the next iteration, the functional value for the new solution will be immediately calculated without its explicit generation. However, with this approach, the amount of computation on the processor almost doubles, which can be critical for a small network.

In some cases, procedure 1 will be optimal in other cases, procedure 2, which was discussed in detail in [8].

Procedure A3. Its main idea is to most accurately estimate the speed of the procedures A1 and A2 when solving a specific applied problem. Since the execution of one iteration of the algorithm requires, as a rule, less than one thousandth of a second of time, and the processor cache appears after some time and the timer has a non-zero error, a large number of iterations should be used for an accurate estimate.

Step 1. Running the procedure A1 on ten thousand iterations with the measurement of the running time.

Step 2. Running the procedure A2 on ten thousand iterations with the measurement of the running time.

Step 3. Based on the results of the execution time, a training procedure is selected for the remaining iterations with a shorter running time.

When training neural networks using the annealing method using connected devices, the framework automatically evaluates the power of computing devices using the A3 procedure and selects the most efficient parallelization option.

Matrix multiplication on the video card is implemented using two-level parallelization. The first level cyclically distributes the calculations of individual blocks of the resulting matrix between the working groups of the video card. The second level cyclically distributes the calculations of individual elements of the block of the resulting matrix between the cores of a separate group. Since the resulting matrix contains a large number of blocks, there is no problem of irregular loading of workgroups. The number of elements in the block is a multiple of the number of cores in the group, so there is no irregular cores loading.

Computing the value of the objective function requires assembling the result on the processor. The calculation of the objective function value is carried out in several stages. At the first stage, shared video memory is created for individual workgroups. At the second stage, each core of the video card calculates a fragment of the objective function value. At the third stage, one core in each working group summarizes the results of the calculations of the cores of the group into memory shared between the groups. Exactly one shared memory cell is allocated for each group. At the fourth stage, the video card transfers the contents of the shared memory to the processor upon completion of work. At the fifth stage, the processor summarizes the results of the groups, because on a video card, synchronization of calculations is possible only within the same working group. The number of working groups is chosen in such a way that there is not too much data transfer and long assembly, and at the same time there is no too long calculation of objective function fragments on the video card.

VIII. Experiments

Let's check the developed software package efficiency using the example of solving the color image compression problem.

For the experiments, the STL-10 dataset from Stanford University was used [9].

The dataset contains 100,000 color images with a resolution of 96x96 pixels. The images can show an arbitrary object [10], which makes compressing image data quite a challenge.

For experiments, 8-fold, 16-fold and 32-fold compressions were chosen. Lower compression ratios are more efficiently produced using classical compression algorithms, and higher ones are meaningless due to too large losses.

For all degrees of compression, the images were divided into fragments of 4 by 4 pixels. Splitting into smaller fragments leads to a decrease in the compression quality, an increase, in turn, leads to an oversized neural network architecture and requires too much data and computing resources for training. Each individual fragment is compressed by a separate restricted Gauss-Bernoulli Boltzmann machine. For 8-fold compression, the number of neurons in the hidden layer of each machine was 48, for 16-fold - 24, for 32-fold - 24, but to achieve the required degree of compression, another layer of restricted Bernoulli-Bernoulli type Boltzmann machines was added with 48 neurons in the input layer and 24 in the hidden layer.

To train restricted Boltzmann machines with the gradient method, the CD-1 algorithm will be used. The PCD algorithm is more efficient on a small number of iterations [11], however, it is based on the assumption captionCompression results 3 bit per pixel

Training algorithm	ADAM	FTML	genetic	annealing
MSE	272	254	322	262
PSNR	23.9	24.2	23.1	24.0
PSNR_HVS	24.1	24.4	23.3	24.1
SSIM	0.746	0.756	0.698	0.733
Training time, h	10	10	30	30

captionCompression results 1.5 bit per pixel

Training algorithm	ADAM	FTML	genetic	annealing
MSE	433	397	452	390
PSNR	21.9	22.3	21.7	22.3
PSNR_HVS	22.1	22.5	21.9	22.5
SSIM	0.663	0.673	0.638	0.669
Training time, h	4	4	18	22

captionCompression results 0.75 bit per pixel

Training algorithm	ADAM	FTML	genetic	annealing
MSE	836	756	697	640
PSNR	19.0	19.4	19.8	20.2
PSNR_HVS	19.2	19.5	20.0	20.3
SSIM	0.502	0.509	0.525	0.551
Training time, h	6	6	21	25

that at a separate iteration the parameters of the network being trained don't change significantly, which doesn't correspond to the problem being solved. The CD-k algorithm requires k times more calculations per iteration than CD-1 and at the same time achieves better quality [12], however, in the problem being solved, the values of the gradients are very large and the use of the CD-k algorithm is not advisable.

To compare the effectiveness of the developed learning algorithms in the framework with existing analogues, the strongest optimization algorithms implemented in analogues were taken: the adaptive moment method [13] (ADAM) due to its stability used for training neural networks of complex architecture [14] and following the moving leader method [15] (FTML), and from the current framework — the original annealing method and genetic algorithm.

For training, the first 8000 images were used as a training set, the next 7000 images for the validation set, the remaining 85000 images formed a test set.

To evaluate the compression efficiency, the most common quality functionals were chosen: MSE (mean squared error), PSNR (peak signal to noise ratio), PSNR-HVS (PSNR human visual system), SSIM (structure similarity image measurement).

The experiments were held on the operating system Lubuntu 20.04 with 4-core CPU intel i7-4770k, 16 GB 1600 MHz RAM and GPU nvidia rtx 3070 with 5888 cores. Compiler version gcc 9.4.0, GPU driver version 470.161.03. The framework has been configured to use the GPU for training.

Compilation options "gcc -xc++ -Wl,-z,stack-size=10000000000 superOpenCLFramework.cpp constants.cpp trainPause.cpp deviceConvLayer.cpp devicePoolingLayer.cpp deviceDeepNN.cpp trainDCNN.cpp poolingLayer.cpp convLayer.cpp mlp.cpp autoencoder.cpp testCompression.cpp deviceMLP.cpp RBMGaussBernoulli.cpp deviceRBMGaussBernoulli.cpp deepNN.cpp trainDeepNN.cpp deepNNFunctioning.cpp dataProcessing.cpp deviceData.cpp superFrameworkInit.cpp finalTrainStats.cpp finetuning.cpp trainMLP.cpp trainRBM.cpp fastMath.cpp trainSettings.cpp -lstdc++ -D_FORCE_INLINES -O3 -l OpenCL -lgomp -lm -fopenmp"

The experiments results are displayed by data compression ratio (see Table 1, 2, 3).

From the experimental results, it can be noted that at low degrees of compression, the annealing method is approximately equal to the gradient methods in terms of the quality of training, but it lags behind them in terms of speed by about 4 times. However, as the compression task becomes more complex, the annealing method begins to surpass them in quality. With 32-fold compression, both random search algorithms significantly outperformed the gradient methods in quality, and the annealing method significantly outperformed the genetic algorithm in the resulting solution quality.

The higher the complexity of the problem being solved, the worse the solution is obtained by gradients compared to random search algorithms. Modern training neural networks frameworks either do not support this type of training algorithms in principle, or their functionality in this part is extremely poor, which makes them not the most effective means of solving complex applied problems.

IX. Conclusion

The paper presents a software package for training neural networks using random search algorithms.

High performance and cross-platform is achieved through the use of OpenMP, OpenCL libraries. A wide supported architectures library in the framework allows us to construct deep neural networks of various architectures, which makes it flexible in applied problems solving. The framework supports a wide variety of computing devices and is able to adapt to low-power computing devices, which makes it possible to use it on a wide variety of devices.

Thanks to the random search algorithms support, the framework is able to solve complex applied problems more efficiently than those existing on gradient methods. Using the example of solving the color image compression problem, it was shown that the proposed framework solves neural networks complex training problem more efficiently than existing analogues. From this we can conclude that the developed software package can be used instead of analogues and has a great development prospect in the future.

References

[1] A. Prieto, B. Prieto, E. M. Ortigosa, E. Ros, F. Pelayo, J. Ortega, and I. Rojas, “Neural networks: an overview of early research, current frameworks and new challenges,” *Neurocomputing*, pp. 242–268, 2016.

[2] M. Kovalev, A. Kroshchanka, and V. Golovko, “Convergence and integration of artificial neural networks with knowledge bases in next-generation intelligent computer systems,” *Otkrytye semanticheskie tehnologii proektirovanija intellektual’nyh sistem [Open semantic technologies for intelligent systems]*, pp. 173–186, 2022.

[3] A. Kroshchanka, “Deep neural networks application in next-generation intelligent computer systems,” *Otkrytye semanticheskie tehnologii proektirovanija intellektual’nyh sistem [Open semantic technologies for intelligent systems]*, pp. 187–194, 2022.

[4] M. Knop, T. Kapuscinski, W. K. Mleczko, and R. Angruk, “Neural video compression based on rbm scene change detection algorithm,” *International Conference on Artificial Intelligence and Soft Computing*, pp. 660–669, 2016.

[5] M. T. Khanna, C. Ralekar, A. Goel, S. Chaudhury, and B. Lall, “Memorability-based image compression,” *IET Image Process*, pp. 1490–1501, 2019.

[6] B. Sharma, M. Tomer, and K. Kriti, “Extractive text summarization using f-rbm,” *Journal of statistics and management systems*, p. 1093–1104, 2020.

[7] V. Krasnoproshin and V. Matskevich, “Random search in neural networks training,” in *Proceedings of the 13-th International Conference “Computer Data Analysis and Modeling” – CDAM’2022*. Minsk : BSU, 2022, pp. 96–99.

[8] V. Matskevich, “Obuchenie neironnykh setei na osnove metoda otzhiga [neural networks training based on annealing method],” *Vestnik Polotskogo gosudarstvennogo universiteta [Bulletin of Polotsk state university]*, p. 21–29, 2022.

[9] Stl-10 dataset. Mode of access: academictorrents.com/details/a799a2845ac29a66c07cf74e2a2838b6c5698a6a. — Date of access: 25.02.2023.

[10] Stl-10 dataset description. Mode of access: stanford.edu/~acoates/ /stl10. — Date of access: 24.02.2023.

[11] K. Oswin, A. Fischer, and C. Igel, “Population-contrastive-divergence: Does consistency help with rbm training?” *Pattern Recognition Letters*, pp. 1–7, 2018.

[12] K. Brugge, A. Fischer, and C. Igel, “The flip-the-state transition operator for restricted boltzmann machines,” *Machine Learning*, p. 53–69, 2013.

[13] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” in *Proc. of the 3rd Intern. Conf. on Learning Representations (ICLR 2015)*, 2015, p. 1–15.

[14] S. Hamis, T. Zaharia, and O. Rousseau, “Image compression at very low bitrate based on deep learned super-resolution,” *IEEE 23rd Intern. Symposium on Consumer Technologies (ISCT)*, p. 128–133, 2019.

[15] S. Zheng and J. T. Kwok, “Follow the moving leader in deep learning,” in *Proc. of the 34-th International Conference on Machine Learning*, 2017, p. 4110–4119.

Нейросетевая программная технология, обучаемая на принципах случайного поиска

Краснопрошин В. В., Мацкевич В. В.

В работе рассматривается актуальная прикладная проблема, связанная с программной реализацией нейросетевой технологии, в рамках которой процесс обучения основан на алгоритмах случайного поиска.

Обучение нейронных сетей является типичной задачей оптимизации. На начальном этапе развития нейросетевых технологий для решения таких задач традиционно использовались различные варианты градиентных методов. Такие методы, как правило, удовлетворяли требования к задаче по качеству и скорости обучения. Однако с появлением нового класса прикладных задач ситуация изменилась. Традиционный подход к обучению с использованием градиентных методов не всегда соответствовал требованиям прикладной задачи по качеству получаемого решения.

В работе предлагается один из вариантов программной реализации нейросетевой технологии (в виде фреймворка) по стандарту ostis 2021, в которой для обучения нейронных сетей используются алгоритмы случайного поиска.

Received 18.03.2023