Neural network software technology trainable on the random search and gradient descent principles

Vadim V. Matskevich
Department of Information and Management Systems
Belarusian State University, Faculty of Applied Mathematics and Informatics
Minsk, Belarus
matskevich1997@gmail.com

Xi Zhou
CETC LES Information System Co., Ltd
China
zhouxi_nju@126.com

Qing Bu
Les International (MSK) Information Technology Co., Ltd
China
39020765@qq.com

***Abstract*—The paper considers an applied problem related to the construction of efficient neural network technologies implemented in the traditional frameworks' standards. It is shown that the increase in efficiency is achieved due to the additional inclusion in the framework's structure of training algorithms based on the ideas of random search. Original implementations of such algorithms are proposed, with experimental confirmation of their effectiveness. It is shown that in this case not only the solutions' obtained quality increases, but it is also possible to extend the range of applied problems to be solved.**

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

I. Introduction

Today, neural network technologies are actively used to solve a wide range of applied problems. However, for their application, it is necessary to pre-configure (train) the neural network's architecture for the problem being solved.

To obtain a high-quality solution, training requires large amounts of data and computational costs. At present, gradient descent algorithms are usually used for training. However, their fast convergence doesn't always guarantee an acceptable quality of the resulting solution. In addition, gradient descent algorithms are very sensitive to the objective function smoothness. Therefore, in recent years, the popularity of training algorithms based on random search has been growing.

Random search algorithms are often used to improve an already obtained solution [1, 2]. They are also used in solving more complex problems, for example, to optimize the trainable neural network's architecture [3, 4, 5, 6, 7]. In addition, with a very limited training time, there are options when gradient algorithms are used to improve the training results by random search algorithms. [8]. However, it is not uncommon for random search algorithms to be used directly to train neural networks [9, 10, 11, 12].

It should be noted that the random search algorithms' implementation in modern frameworks is extremely rare. Their implementation is limited, as a rule, by genetic algorithms. Thus, the training neural networks' problem is still relevant.

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

II. Problem Formulation

A wide range of applied problems is solved using neural network technologies implemented in the form of frameworks.

Today, there are a number of frameworks for solving machine learning problems. The most popular among them are MXNet, PyTorch, Tensorflow 2 and Caffe 2.

1. MXNet is implemented in C++. It is a high-performance, cross-platform framework and contains the following set of optimizers: AdaDelta, AdaGrad, ADAM, DCASGD, FTML, FTRL, LARS, LBSGD, NAG, Nadam, RMSProp, SGD, LAMB. The framework supports a wide class of neural network architectures. It is designed to train neural networks with a large number of configurable parameters on computers with high computational power.

Among the shortcomings, one can note a not very convenient (compared to simpler frameworks) user interface and a restricted class of optimization algorithms. All of them are different modifications of gradient descent. At the same time, there is no support for random search methods.

2. PyTorch is implemented in Python. The framework is cross-platform and supports training on video cards. It contains the following set of optimizers: Adadelta, Adagrad, Adam, AdamW, SparseAdam, Adamax, ASGD, LBFGS, NAdam, RAdam, RMSprop, Rprop, SGD. The framework supports a wide class of neural network architectures, has a native user-friendly interface in Python.

Among the shortcomings are the following: due to the costs of the Python programming language, the performance of training process is not high enough.

3. Tensorflow 2 implemented mostly in the language Python. Separate modules are implemented in the C++ (examples are modules for interaction with microcontrollers). This framework is cross-platform and has a simple user interface. It is the most common framework for solving applied problems. It contains the following set of optimizers: Adadelta, Adafactor, Adagrad, ADAM, AdamW, Adamax, FTRL, Nadam, RMSprop, SGD, differentialEvolution. It supports a wide class of neural network architectures and has a native user-friendly interface in Python.

The disadvantages of the framework include insufficient high performance (costs of the Python programming language) and a restricted set of non-directional optimization algorithms.

4. Caffe 2 is implemented in C++. It is a high-performance and cross-platform framework. It has the following set of optimizers: Adadelta, Adagrad, ADAM, FTRL, GFTRL, RMSprop, SGD, YellowFin. The framework lacks of support for recurrent neural networks.

Among the disadvantages are the limited classes of training algorithms and neural network architectures. In addition, the lack of support for recurrent neural networks makes it impossible to use the framework for solving applied problems with non-static data (text processing, speech recognition and video sequence processing).

Thus, modern frameworks, for all their popularity, are either limited by the set of training algorithms and support a restricted class of neural networks, or they do not have high performance.

III. Framework Architecture and Composition

The software package [13] was developed in C++ using the OpenMP and OpenCL libraries. This ensures its high performance and cross-platform.

The framework consists of the following main modules: libraries of training algorithms and neural network architectures, configuration files with algorithm parameters, modules for execution on various computing devices, and a user interface.

The algorithms library contains a wide range of optimization algorithms: gradient training algorithms (SGD, MomGrad, ADAM, FTML), algorithms based on random search (Slow annealing, batch annealing, genetic, batch genetic) and hybrid training algorithms (greedy, batch greedy).

The framework 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.

The framework can run on a limited number of CPU and GPU threads (limitations can be adjusted). In addition, a completely single-threaded execution mode is possible.

The user interface contains the following functionality: loading (saving) a neural network from a hard disk, constructing neural networks based on architectures base, training neural networks, compressing and decompressing bitmaps, classifying objects.

IV. Framework Using Features

The framework operates in seven main modes: loading neural networks from hard disk, saving neural networks to hard disk, constructing deep neural networks, setting parameters and invoking the neural networks training process, loading input data, compressing and recovering color bitmaps, and classifying images.

The neural network is loaded from the hard disk by calling the appropriate function. Its implementation is quite trivial and does not require additional comments.

The neural network is saved to the hard disk when the user calls the corresponding function. It should be noted that the format of the saved network corresponds to the format when the network is loaded from disk.

The loading of input data is presented in the framework is rather limited. When developing the framework, only the rgb format of input images was implemented, the user must additionally specify the number of images and their resolution.

Compression and decompression of color bitmaps is performed by calling the corresponding functions. They take a deep neural network and input data as parameters.

The classification of images is done by calling the predict function. It takes as parameters a deep neural network, input data and information about their amount. This function returns a numeric array - numbers of classes to which the input images belong. It is assumed that the user knows which numbers correspond to which classes.

The user can create a neural network by sequentially connecting the layers of the network being designed. After building the network, the framework checks the correctness of the built architecture.

For training, the user, in addition to input data loading and neural network designing, must set the training parameters.

For each network's layer, it is necessary to define: the time allocated for training, the training algorithm, the need to continue training or lack of layer's training, the amount of data for the validation and training sets, and the need to take into account the results on the validation set in the training stop criterion.

After that, the user sets the general training settings: the need to use the processor for training, forming and converting input images, the number of processor and video cards threads used. Other video card settings are calculated automatically, but the user can change them manually.

After the settings, the user calls the neural network training function, which automatically connects all the necessary devices and modules for training, and also loads the training algorithms parameters.

At the end of training, the neural network is saved to the hard drive of the computer..

V. Neural Networs Hybrid Training Algorithm

Random search algorithms are characterized by a large solution search space. This leads to the need (to achieve an acceptable solution) to perform a large number of iterations. Gradient-type algorithms, in turn, due to the strict transition rule, overly limit the search space and thus may potentially miss the optimal solution.

The idea of building a hybrid algorithm is to expand the search space compared to directed methods and reduce the search space compared to random search methods. The developed algorithm consists of the following steps:

Preliminary step. Initialization (randomly) of the initial solution x_0 and $f(x_0)$ calculation.

General *k*-th iteration.

Step 1. Генерация случайного решения y на основе текущего решения x и вычисление $f(y)$. Данный шаг полностью аналогичен генерации в алгоритме на основе метода отжига.

Step 2. Current solution x is replaced to y, if $f(x) \geq f(y)$

Step 3. Checking stop criteria. If for N successive perfect iterations (N is algorithm's parameter) there were not transitions to a new solution, then the algorithm ends. Otherwise, the transition to the next iteration is performed.

With a help of procedure for a random solution generation it is possible to tune the size of the solution space. When generating a wide vicinity of the current solution, it is possible to generate almost any solution in several iterations. Given an ε-vicinity of the current solution, the algorithm degenerates into a simple gradient method.

Thus, the described algorithm is a kind of compromise between the solution quality and speed.

This algorithm does not guarantee the achievement of an optimal solution. If the generation of new solutions in a small vicinity doesn't lead to an acceptable solution, then it makes no sense to apply the algorithm. As the vicinity expands, the convergence slows down significantly, and it is preferable to use the annealing algorithm to ensure that the optimal solution is reached. Narrowing the vicinity does not make sense, because this further limits the search space for a solution.

VI. Scalable Modification of Random Search Algorithms

To solve the scalability problem, the following modification of random search algorithms (batch annealing, batch genetic, batch greedy) is proposed. In this case, scalability means the amount of computations independence from the amount of input data.

At the initialization of initial solution stage, the training dataset is duplicated *Q* times, where *Q* is algorithm's parameter. After that, an elements random permutation is performed within each original dataset's copy. Dataset's duplication and permutation of elements within the copies increase the dataset's fragments diversity, which improves the training quality in general. The dataset increased in this way is divided into *QM* fragments, where *M* is algorithm's parameter. It is assumed that the training dataset is divided into fragments without a remainder. The parameter *M* is selected in such a way that the dataset is divided into big data fragments.

Splitting into small fragments leads to low accuracy of the objective function estimation on the entire training set and poor training quality. On the other hand, splitting the dataset into too large fragments requires an excessively large number of iterations and calculations to ensure convergence.

At the initialization stage, the objective function value is accurately calculated on the entire training set. To do this, all fragments of the training dataset are fed into the network being trained one by one and the objective function values of are calculated. The calculated values for each fragment are stored, summed up - this is the exact objective function value multiplied by *Q*.

At each iteration of this algorithm, value is optimized on one of the fragments of the training set. At the first iteration, the objective function value for the first fragment is calculated. Every *K* iterations, a cyclic change of training dataset's fragment is performed, where *K* is algorithm's parameter.

At each iteration, the objective function value multiplied by *Q* is estimated on the entire training set. To do this, its estimate is calculated from the sum of the old values for all its fragments. The new value of the objective function is defined as the subtraction from old estimate the old value for the current fragment and add the objective function value on this fragment for the new solution.

Thus, the procedure described above completely solves the problem of training algorithm scaling. As in the case of gradient algorithms, the size of the training dataset fragment at each iteration is a constant that does not depend on the size of the network being trained. This provides a linear increase in the training complexity with the network size growth.

VII. Experimental Results

For the experiments, the STL-10 dataset was used [14, 15]. For experiments, 8x, 16x, and 32x compressions were chosen. For all degrees of compression, the images were divided into fragments of 4 by 4 pixels. Each separate fragment is compressed by a separate restricted Gauss-Bernoulli Boltzmann machine. The following architectures were used for compression: for 8x 48-48, for 16x 48-24, for 32x 48-24 48-24. For 32-fold compression, the second layer was a restricted Bernoulli-Bernoulli type Boltzmann machine.

To test the effectiveness of the developed framework, the most popular optimization algorithms were taken: adaptive moment method (ADAM) [16] and following the moving leader method (FTML) [17].

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 with human visual system), SSIM (structure similarity image measure).

The experiments were carried out on the Lubuntu 20.04 operating system using an nvidia rtx 3070 video card (see Table I, II, III).

TABLE I. COMPRESSION RESULTS FOR 3 BITS PER PIXEL

Training algorithm	**Quality function**				
	MSE	***PSNR***	***PSNR_HVS***	***SSIM***	***Training time, h***
ADAM	272	23.9	24.1	0.746	10.0
FTML	254	24.2	24.4	0.756	10.0
annealing	262	24.0	24.1	0.733	30.0
batch annelaing	254	24.1	24.3	0.733	30.0
genetic	322	23.1	23.3	0.698	30.0
batch genetic	315	23.2	23.4	0.692	15.0
greedy	306	23.4	23.6	0.723	10.0
batch greedy	271	23.9	24.1	0.737	10.0

TABLE II. COMPRESSION RESULTS FOR 1.5 BITS PER PIXEL

Training algorithm	**Quality function**				
	MSE	***PSNR***	***PSNR_HVS***	***SSIM***	***Training time, h***
ADAM	433	21.9	22.1	0.663	4.00
FTML	397	22.3	22.5	0.673	4.00
annealing	390	22.3	22.5	0.669	22.0
batch annelaing	384	22.4	22.6	0.671	22.0
genetic	452	21.7	21.9	0.638	18.0
batch genetic	413	22.1	22.2	0.644	12.0
greedy	415	22.1	22.3	0.661	10.0
batch greedy	385	22.4	22.5	0.671	10.0

TABLE III. COMPRESSION RESULTS FOR 0.75 BITS PER PIXEL

Training algorithm	Quality function				
	MSE	***PSNR***	***PSNR_HVS***	***SSIM***	***Training time, h***
ADAM	836	19.0	19.2	0.502	6.00
FTML	756	19.4	19.5	0.509	6.00
annealing	640	20.2	20.3	0.551	25.0
batch annelaing	632	20.2	20.4	0.557	28.0
genetic	697	19.8	20.0	0.525	21.0
batch genetic	718	19.7	19.9	0.524	11.5
greedy	707	19.7	19.9	0.524	11.5
batch greedy	692	19.8	19.9	0.534	13.0

From the experimental results, it can be noted that at low compression ratios, the most efficient random search algorithms are approximately equal in obtained solution quality, but more than 4 times slower than it. Scalable modification of all random search algorithms either significantly reduces training time or improves the quality of the resulting solution.

The theoretical guarantee of the convergence of the annealing method to the optimal solution allowed the algorithms built on its basis to achieve the highest quality of the obtained solution (among the presented algorithms). However, it was also the slowest.

At high compression ratios (32 times and higher), random search algorithms are significantly superior to gradient algorithms in terms of the solution obtained quality. At the same time, the quality of the solution obtained by the hybrid algorithm is lower than that of random search algorithms, but significantly higher than that of gradient algorithms. The hybrid training algorithm turned out to be 2.5 times faster than random search algorithms. This result shows that it is a full of value compromise between the high obtained solution quality and the high training speed.

VIII. CONCLUSION

The paper presents a framework in which algorithms based on random search are used to train neural networks.

Due to the use of the OpenCL and OpenMP libraries in the framework development, its high performance and cross-platform were ensured. This naturally expands the possibilities of its application for training neural networks.

It has been experimentally shown that the training algorithms implemented in the framework based on random search provide a higher quality of the obtained solution. The framework also implements gradient training algorithms for the case of extreme time limitations for training.

Thus, the framework proposed in the work expands the range of existing analogues and has a great development prospect in the future.

It was used to train a neural network classifier when building a system for detecting non-weather changes in the landscape. Due to the presence of random search algorithms for neural networks training, a high efficiency of the system was obtained.

REFERENCES

[1] Rehman, S., Nuha, H. H., Al Shaikhi, A., Akbar, S. & Mohandes, M. Improving Performance of Recurrent Neural Networks Using Simulated Annealing for Vertical Wind Speed Estimation. Energy Engineering Vol.120, No.4, 2023. pp. 775–789. DOI: 10.32604/ee.2023.026185.

[2] Ying Yan, Wenting Zhang, Yongzhi Liu, Zhixuan Li Simulated annealing algorithm optimized GRU neural network for urban rainfall-inundation prediction. Journal of Hydroinformatics Vol.25, No.4, 2023: pp. 1358–1379. DOI: 10.2166/hydro.2023.006.

[3] Kuo, C. L., Kuruoglu, E. E. & Chan, W. K. V. Neural network structure optimization by simulated annealing. Entropy, Vol.24, No.3, 2022. pp. 348–365. DOI: 10.3390/e24030348.

[4] Tsai, C.W., Hsia, C.H., Yang, S.J., Liu, S.J., & Fang, Z.Y. Optimizing hyperparameters of deep learning in predicting bus passengers based on simulated annealing. Applied Soft Computing, Vol.88, No.3, 2020. pp.106068–106076. DOI: 10.1016/j.asoc.2020.106068.

[5] Deng, W., Liu, H., Xu, J., Zhao, H. & Song, Y. An improved quantum-inspired differential evolution algorithm for deep belief network. IEEE Transactions on Instrumentation and Measurement, Vol.69, No.10, 2020. pp. 7319–7327. DOI: 10.1109/TIM.2020.2983233.

[6] Hamdia, K.M., Zhuang, X. & Rabczuk, T. An efficient optimization approach for designing machine learning models based on genetic algorithm. Neural Computing & Applications. Vol.33, 2021. pp. 1923–1933. DOI: 10.1007/s00521-020-05035-x.

[7] Al Haromainy, M. M., Prasetya, D. A. & Sari, A. P. Improving Performance of RNN-Based Models With Genetic Algorithm Optimization For Time Series Data. TIERS Information Technology Journal, Vol.4, No.1, 2023. pp. 16–24. DOI: 10.38043/tiers.v4i1.4326.

[8] Liu, Y., Cuiqing J., Cuiping L., Zhao W., Wanliu Ch. Increasing the Accuracy of Soil Nutrient Prediction by Improving Genetic Algorithm Backpropagation Neural Networks, *Symmetry* Vol.15, No.1, 2023. pp. 151–165. DOI: 10.3390/sym15010151.

[9] He, F., Ye, Q. A Bearing Fault Diagnosis Method Based on Wavelet Packet Transform and Convolutional Neural Network Optimized by Simulated Annealing Algorithm. *Sensors*, Vol.*22*, 2022. pp. 1410–1426. DOI: 10.3390/s22041410.

[10] Osegi, E. N. & Jumbo, E. F. Comparative analysis of credit card fraud detection in Simulated Annealing trained Artificial Neural Network and Hierarchical Temporal Memory. Machine Learning with Applications, Vol.6, 2021. pp. 100080–100089.

[11] Chakravorty S. & Nagarur N. N. Simulated Annealing Based Artificial Neural Network For Real Time Dispatching, *34th Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, 2023. pp. 1–6. DOI: 10.1109/ASMC57536.2023.10121138.

[12] Gao Y., Liu L., Chen M. & Tian L. Mechanical properties analysis of cell surface based on genetic simulated annealing optimization neural network. Proceedings of International Conference on Internet of Things and Machine Learning (IoTML), 2022. pp. 316–324. DOI: 10.1117/12.2673514.

[13] Krasnoproshin, V. V. & Matskevich, V. V. Neural network software technology trainable on the random search principles, Research Papers Collection “Open Semantic Technologies for Intelligent Systems”, Vol. 7, 2023. pp. 133–140

[14] STL-10 dataset. – link: academictorrents.com/details/a799a2845ac29a66c07cf74e2a2838b6c5698a6a – Access date : 25.02.2023.

[15] STL-10 dataset description. – link: stanford.edu/~acoates//stl10/ – Access date : 24.02.2023.

[16] Kingma, D. P. & Ba J.L. Adam: A Method for Stochastic Optimization, Proc. of the 3rd Intern. Conf. on Learning Representations (ICLR), 2015. pp. 1–15.

[17] Zheng Sh. & Kwok J. T. Follow the moving leader in deep learning, Proc. of the 34-th International Conference on Machine Learning, 2017. Vol. 70, 2017. pp. 4110–4119.