Fast Random Search Algorithm in Neural networks Training

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

***Abstract*—The paper deals with a state-of-art applied problem related to the neural networks training. Currently, gradient descent algorithms are widely used for training. Despite their high convergence rate, they have a number of disadvantages, which, with the expansion of the neural networks' scope, can turn out to be critical.**

The paper proposes a fast algorithm for neural networks training based on random search. It has been experimentally shown that in terms of the proposed algorithm's convergence rate, it is almost comparable to the best of the gradient algorithms, and in terms of quality it is significantly ahead of it.

Keywords—neural networks, random search, gradient decent, training.

I. Introduction

Neural networks are at the heart of many modern information systems. With the digital devices development, the processed data amount is constantly growing. This leads to the need to design neural networks with a large number of tunable parameters. Setting parameters for a specific application task is based on the training process. The efficiency of information systems as whole depends on the neural networks training quality. Therefore, the relevance of training neural networks training in the face of increasing complexity of their architecture is constantly increasing.

Gradient descent algorithms are traditionally used to train neural networks [1]. However, having fast convergence, they do not always guarantee the resulting solution quality. To solve this problem, training algorithms based on random search are used [2]. With the growth of modern computers' computing power, the popularity of such algorithms is constantly growing.

II. Problem Analysis

Training algorithms based on the idea of gradient descent have become widespread due to their high convergence rate [3]. However, they require objective function differentiability, which significantly narrows the class of applied problems to be solved. Moreover, algorithms of this type can converge to solutions where the gradient value is close to or equal to zero [4]. This, in turn, can lead to a decrease in the resulting solution quality.

To combat this problem, training algorithms based on random search began to be used. The most common among them are genetic algorithms and annealing algorithms [5]. However, this class of algorithms is not widely used due to the low convergence rate. Consider convergence issues in more detail.

Gradient descent methods train neural networks quickly due to the small number of iterations required to ensure convergence. In practice, the convergence of such algorithms requires about 10^5 iterations, while for the random search method – about 10^6.

In addition, gradient descent methods have the scalability property. When training neural networks with a large number of tunable parameters, a large training set is required. The gradient descent method at each iteration calculates the gradient value and updates the parameters based on a fairly small fragment of the training dataset. The random search method, in turn, at each iteration calculates the objective function value on the entire dataset. This leads to a quadratic (compared to linear for the gradient) increase in the amount of computation at a single iteration.

Thus, it is necessary to develop a random search algorithm that in practice would have an acceptable convergence rate.

III. Training Algorithm

To train neural networks, a hybrid algorithm that uses the ideas of random search and gradient descent is proposed. In this algorithm, to construct variants of transition to a new solution, the current solution's vicinity is randomly generated, and the transition is carried out only to the solution for which the error functional value doesn't increase.

Consider the problem of some objective function f minimizing. We will assume that the optimized parameters can be divided into sets, each of which has its own supposed optimal range of values. The algorithm consists of the following steps.

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

Step 1. New solution y generation and $f(y)$ calculation.

Step 1.1 *Generating random variables procedure*.

m uniformly distributed on a segment from zero to a number equal to the number of parameters in the set random variables $a_1,a_2,\ldots,a_m$ are generated. m random permutations of length equal to the number in the parameter set are generated. The first $a_1,a_2,\ldots,a_m$ of elements of each permutation define the parameters' indexes to be changed in each set of parameters, respectively.

Step 1.2 *Generating new solution procedure*.

For each changeable parameter uniformly distributed on a segment $[-l/2;l/2]$ random variable b is generated. The value of l depends on which set the variable parameter belongs to, and is equal to l_1, l_2, …, l_m respectively. The l values for each set are given as algorithm's parameters.

Let x_i be a changeable parameter, and it's new value be x_i', then: $x_{i'}=x_i+b$.

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

Step 3. 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 go to Step 1.

To solve the scalability problem, the following algorithm's modification is proposed (see Fig. 1). In this case, scalability means the amount of computations independence from the amount of input data.

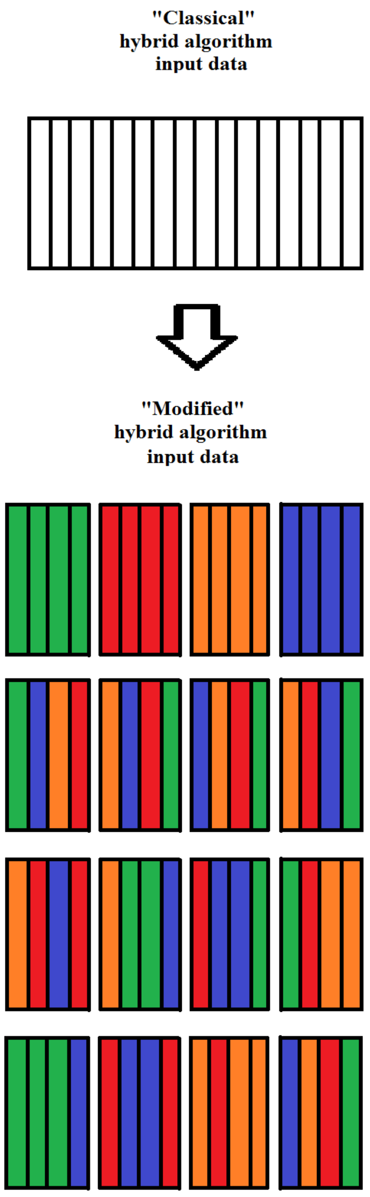
"Classical" hybrid algorithm input data
"Modified" hybrid algorithm input data

Fig. 1. Hybrid algorithm data mixture.

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.

The proposed training algorithm completely solves the scaling problem. 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.

Using the random solution generation procedure, one can control the solution space size. When generating a wide current solution's vicinity, it is possible to generate almost any solution in several iterations. If for generation we specify the current solution's ε-vicinity, then the algorithm degenerates into a simple gradient method.

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

IV. Experiments

To compare the developed algorithm efficiency with gradient methods, let's consider solving the problems of color image compression and pattern recognition. For comparison with gradient methods, the best training algorithm will be used – following the moving leader (FTML) [6].

For experiments CIFAR-10 [7] and STL-10 [8, 9] datasets were used. The STL-10 sample was used to compress color images with a resolution of 96x96, and CIFAR-10 was used to pattern recognition on color images with a resolution of 32x32.

To classify images, not very difficult separable classes were used to simplify the training algorithms effectiveness comparison. The most common quality functionals were used to evaluate the compression quality: mean squared error (MSE), peak signal to noise ratio (PSNR), with human visual system (PSNR-HVS), structural similarity image measure (SSIM).

For the experiments, 8-fold, 16-fold and 32-fold compression were chosen. A higher compression ratio results in too much loss. At the same time, a lower compression ratio does not make sense due to the presence of efficient classical compression algorithms.

To compress color images, a deep belief network was designed based on restricted Boltzmann machines. For 8-fold compression, the images were divided into blocks of 4x4 pixels. Each image block was used to train a separate Gauss-Bernoulli type restricted Boltzmann machine. Thus, for compression, a single-layer neural network was constructed from a composition of 576 restricted Boltzmann machines of the same architecture. The input layer of each machine had 48 neurons, while the hidden layer had 48 neurons. This provides 8-fold compression. For 16-fold compression in machines, the number of neurons in the hidden layer was reduced to 24. For 32-fold compression, a two-layer belief network was constructed. The first layer had the same architecture as with 16-fold compression. The second layer was a composition of 288 restricted Bernoulli-Bernoulli-type Boltzmann machines. There are 48 neurons in the input layer of each machine, and 24 in the hidden layer.

For pattern recognition, a deep belief network was constructed based on autoencoders with a classifier in the last layers in the form of a multilayer perceptron. The images were divided into blocks of 2x2 pixels. A separate autoencoder was trained for each block The first layer of the network was a composition of 256 autoencoders of the same architecture. There are 12 neurons in the autoencoder input layer, and 6 in the hidden layer. The second layer of the network consisted of a composition of 128 autoencoders of the same architecture. After that, a multilayer perceptron is located in the neural network. The input layer contains 768 neurons. The hidden layer contains 8 neurons with a bipolar sigmoid activation function. The output layer consists of two neurons with a softmax activation function.

Training was carried out on a computer with a video card nvidia rtx 3070 with 5888 cores (driver version 470.161.03) and 4-core CPU intel i7-4770k with 2x8 GB DDR3 1600MHz RAM on operating system Lubuntu 20.04. The time was measured by calling the gettimeofday function. The training algorithms are implemented in a special cross-platform framework [10] using the OpenMP and OpenCL libraries in C++.

The obtained experiments results show the proposed training algorithm high efficiency (see Table I, II).

TABLE I. IMAGE COMRESSION RELUTS

Training algorithm	Compr. ratio, bit/pix.	MSE	PSNR	PSNR_ HVS	SSIM	Train. time, h
FTML	3	254	24.2	24.4	0.756	10.0
	1.5	397	22.3	22.5	0.673	4.00
	0.75	756	19.4	19.5	0.509	6.00
Proposed	3	271	23.9	24.1	0.737	10.0
	1.5	385	22.4	22.5	0.671	10.0
	0.75	692	19.8	19.9	0.534	13.0

TABLE II. PATTERN RECOGNITION RESULTS

Classes pairs	6-8	1-4	5-9	0-2
FTML	93.3	89.9	88.2	83.6
Proposed	93.5	89.4	89.0	83.4

From the color image compression results it can be seen that as the problem being solved becomes more complex (see Table 1), the developed algorithm begins to significantly outperform gradient training algorithms. It turned out to be on average 1.9 slower than the gradient, but in this case it is not critical, because training time remains within reasonable limits.

In solving the pattern recognition problem (see Table 2), the proposed algorithm is not inferior in terms of training speed. Moreover, on average, the proposed algorithm shows better results compared to gradients. Achieving a large advantage in this problem is quite difficult due to the dataset complexity.

V. CONCLUSION

The paper proposes a hybrid training algorithm based on combining of random search and gradient descent ideas. It has been experimentally shown that this algorithm is superior in the obtained solution quality to gradient algorithms, but is slightly inferior in training speed, which is not critical.

The proposed in the paper algorithm has no significant limitations in its application in practice. Due to the algorithm scalable modification, it becomes possible to use large training datasets and train large neural networks. Moreover, as the computing power grows, the set of solutions considered by the algorithm increases in the time allocated for neural network training, which leads to obtained solution's quality increase. Thus, we can conclude that the proposed in the paper hybrid training algorithm has a certain promise in applied problems solving.

REFERENCES

[1] D. P. Kingma, J. L. Ba, “Adam: A Method for Stochastic Optimization,” Proc. of the 3rd Intern. Conf. on Learning Representations (ICLR 2015), pp. 1 Journal of Machine Learning Research 23(260). – 2022. – pp. 1-15, 2015. DOI: 10.48550/arXiv.1412.6980.

[2] DW. Zhang, L. Weilin, W. Xiaohua, L. Xiaofeng, “Application of simulated annealing genetic algorithm-optimized back propagation (BP) neural network in fault diagnosis,” International Journal of Modeling Simulation and Scientific Computing 10 (4). 2019. DOI:10.1142/S1793962319500247.

[3] A. Jentzen, A. Riekert, “A proof of convergence for the gradient descent optimization method with random initializations in the training of neural networks with ReLU activation for piecewise linear target functions,” Journal of Machine Learning Research 23(260). 2022.– pp. 1–50.

[4] D. Federici, “Limitations of gradient methods in sequence learning,” Proceedings of the 9th International Conference on Neural Information Processing, 2002. ICONIP'02. – IEEE, 2002. – pp. 2369–2373.

[5] S. Ding, C. Su, J. Yu, “An optimizing BP neural network algorithm based on genetic algorithm,” Artificial intelligence review. – Springer 2011. 36. – pp. 153–162. DOI 10.1007/s10462-011-9208-z.

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

[7] CIFAR-10 dataset – link: https://www.cs.toronto.edu/~kriz/cifar.html – Access date: 15.05.2023.

[8] STL-10 dataset – link: web.archive.org/web/20110803194852/ stanford.edu/~acoates//stl10/ – Access date: 15.05.2023.

[9] STL-10 dataset description. – link: stanford.edu/~acoates//stl10/ – Access date: 15.05.2023.

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