Development of the heuristic method of evolutionary strategies for reinforcement learning problems solving

Maksim Naumov
Samara National Research University
Samara, Russia
blagov@ssau.ru

Aleksandr Blagov
Samara National Research University
Samara, Russia
blagov@ssau.ru

***Abstract*—There are many methods for machine learning problems solving. Each of them is used depending on the solved problems. This article describes several classical algorithms of evolutionary strategies in the reinforcement learning problems. The authors propose the Heuristic method that has certain advantages over existing methods. The article also provides a comparative analysis of the solutions to problems, including the problems of error-correcting coding.**

***Keywords*—reinforcement learning, machine learning, data mining, heuristic method**

I. Introduction

The article describes the research and development of methods of heuristic evolutionary strategies for solving the problem of training recurrent neural networks. Artificial neural networks are one of the most popular methods of machine learning They represent a mathematical model, as well as its software or hardware implementation, built on the principle of the organization and functioning of biological neural networks [1]. The formation and training of neural networks can be reduced to an optimization problem. The authors propose the improvement of the simplest implementation of evolutionary strategies using heuristic methods for training recurrent neural networks in the problem of error-correcting coding

The practical value of improving existing algorithms lies in expanding the range of initial approximations from which the method will converge to a solution. It can also affect the change in the rate of convergence.

Evolutionary strategy is an optimization method based on the ideas of evolution [2]. The relevance of their use is due to the high efficiency of solving optimization problems [3, 4]. Possible solutions are encoded as vectors of real numbers. This method has been successfully used to solve reinforcement learning problems [4-6].

Evolutionary strategies have the following features.

1. They are able to optimize any models that can be expressed through the real numbers vectors (not dependent on optimized model).

2. No calculation of the gradient of parameters of the optimized model required.

3. It is not required to perform the back propagation of gradients, which avoids many problems when multiplying them.

4. They are invariant with respect to the delay before the response of the system.

5. They are invariant to transformations of the space of solutions that preserve angles.

6. They are invariant to transformations of the objective function.

7. A very high degree of concurrency, due to the very small amount of data sent.

There are approaches that use evolutionary strategies not for optimization, but for the numerical calculation of the gradient of the optimized function. Thus, it is possible to use higher-level and more accurate methods, without the need to calculate the gradient of the function.

II. Traditional methods of evolutionary strategies

The main cycle of evolutionary strategy consists of two stages: mutation and selection. Let us consider these stages as an example of maximizing some vector function. $f(x)$:

1. Choose the initial solution. After it make the initial solution for the current.

2. Mutation step: from the current solution we create μ new ones by adding random vectors distributed over the multidimensional normal distribution to the current solution μ with the expectation equal to the current solution and the correlation matrix σI.

3. Calculate the value of the function f in each of the possible solutions.

4. The selection step: as the current solution, we take the best solution possible (or leave the current one if it is better), or calculate the weighted average of the possible solutions, given that more remote solutions have more weight.

5. Repeat steps 2-4 until we get an acceptable solution.

In step 4, the following formula is used

$$\hat{x} = \frac{1}{\mu\sigma}\sum_{k=1}^{\mu}\left[x_k \frac{f(x_k) - \hat{f}}{\tilde{f}}\right], \qquad (1)$$

where $\hat{x}$ is the average solution, x_k is the possible solution numbered k, $\hat{f}$ is the sample mean $f(x_k)$, $\tilde{f}$ is the sample variance $f(x_k)$.

This algorithm does not cover all possible evolutionary strategies. However, it shows their distinguishing features:

- the solution is represented as a vector of real numbers;

- at each step, one (or several) key decision is selected, which is used in the next step as the basis for new solutions;
- selection takes place in a deterministic way, which means that it can be performed on the cluster without sending data;
- the relative simplicity of the algorithm, which is reflected in the speed of the algorithm.

High speed and large parallelism of the algorithm allows for a greater number of iterations compared to other methods that solve reinforcement learning problems.

The above algorithm is also called a simple evolutionary strategy. Due to the fact that does not change the parameter σ depending on the already known data. This impairs convergence, or makes finding a solution impossible.

This algorithm was implemented in this paper. The computational complexity of one iteration is $O(N)$ with a small constant, where N– task dimension.

There are many algorithms to solve the problem of constant parameter σ. This article discusses a method called CMA-ES (covariance matrix adaptation evolution strategy) [7] as the most popular. As an implementation, a library for Python called Pycma is used. Consider a simplified version of this algorithm (reflecting the main ideas of the algorithm), with the dimension of the problem equal to *N*:

1. Choose the number of possible solutions at each step μ. Typically, a value greater than four is selected. Choose an initial solution and make it current (m). Choose the initial parameter vector σ of length *N*, which is responsible for the step length in each direction. Set the correlation matrix C equal to the identity matrix, with $N \times N$ dimension.

2. Generate μ random vectors x_k, corresponding to possible solutions. The generation comes from a multidimensional normal distribution with a correlation matrix *C* and a mathematical expectation of *m*.

3. Calculate the value of the function *f* in each possible solution.

4. Sort possible solutions by the value of the function *f*.

5. Update the value of the correlation matrix *C*, taking as a basis the data on the distribution of possible solutions. The calculations are made according to the formula:

$$\hat{x} = \frac{1}{\mu\sigma}\sum_{k=1}^{\mu}\left[x_k \frac{f(x_k) - \hat{f}}{\hat{f}}\right], \tag{2}$$

where x_k is the k random vector, N_{best} is the number of best solutions to consider; $\hat{x}$ is the average N_{best} vectors, matching the best possible solutions

6. Update the value of the current solution m, according to the formula:

$$C_{ij} = \frac{1}{N_{best}}\sum_{k=1}^{N_{best}}\left(x_{k_i} - \hat{x}_i\right)\left(x_{k_j} - \hat{x}_j\right), \tag{3}$$

7. Repeat steps 2-6 until you get the right solution.

This algorithm performs steps 2-6 in $O(N^2)$ (due to the fact that N_{best} is a constant), where N – task dimension. There are approaches that reduce complexity to $O(N)$with a large constant. This article uses the most complete and correct implementation of the algorithm from the Pycma library for Python.

III. Development and implementation of the method of heuristic evolutionary strategies

The above algorithms were either inaccurate (simple evolutionary strategy) or slow (adaptive evolutionary strategies). Is it possible to build an algorithm with computational complexity $O(N)$ with a small constant and at the same time more accuracy than a simple implementation of evolutionary strategies?

Consider the following modification of a simple evolutionary strategy:

1. Choose the initial solution. After it make the initial solution for the current.

2. Mutation step: from the current solution we create μ new ones by adding random vectors to the current solution μ distributed over a multidimensional normal distribution with a mathematical expectation equal to the current solution and the correlation matrix σI.

3. Calculate the value of the function f in each of the possible solutions.

4. Selection step: as the current solution, we take the best solution possible (or leave the current one if it is better), or calculate the weighted average of the possible solutions, given that more remote solutions have more weight.

5. Calculate the new value of σ using one of the heuristic functions, which will be considered later.

6. Repeat steps 2-5 until we get an acceptable solution.

At step 5, instead of calculating the parameter σ, we can calculate the entire correlation matrix. However, this will increase the computational complexity of the algorithm from $O(N)$ to $O(N^2)$.

Under the heuristic understand the totality of techniques and methods that facilitate and simplify the solution of cognitive, constructive, practical problems.

In this article, heuristic functions are understood to mean functions that improve the accuracy of calculations without increasing the asymptotic complexity.

Consider at some examples of such functions:

$$h_1(fx, i, N) = \frac{1}{1 + e^{-fx}}; \tag{4}$$

$$h_2(fx, i, N) = arctan(fx) + \frac{\pi}{2}; \tag{5}$$

$$h_3(fx, i, N) = \frac{N - i}{N}; \tag{6}$$

$$h_4(fx, i, N) = 1 - sin^2\left(\frac{i}{N}10\pi\right), \tag{7}$$

where (4) - (7) are heuristic functions; fx is the value of the optimized function at the point of the current solution; i is the number of the current iteration; N is the limit number of iterations.

Function (6) implements an idea similar to temperature in the simulated annealing method [8]. At the beginning of optimization, the algorithm considers more distant points, and then gradually "cools down" and proceeds to a more accurate search for a solution. This approach allows the optimization process to go to more promising solutions at the very beginning, and then refine them.

The heuristic function (7) is a rather interesting case. It goes through several full periods during the optimization function. This allows the optimization process to go out of local optima, speeding up the solution.

Not all heuristic functions are similar to those listed, however, this list gives some starting point for finding a suitable kind of function.

From the above functions, you can make a linear combination using positive coefficients and get a new heuristic function.

IV. Experimental comparison of implemented methods

As a problem, on the solution of which the implementations of the above algorithms will be compared, the problem of error-correcting coding was taken.

One of the most famous cyclic codes is the code (7; 4; 3). This code converts messages from k = 4 bits into code words of length n = 7, using for this a generating polynomial of degree r = 3. This code allows us to correct a single error, or detect double errors.

If we assume that the cyclic code (7; 4; 3) is used to correct single errors, then the probability of correctly decoding the message is

$$(1-p)^7 + 7p(1-p)^6. \tag{8}$$

A combination of two LSTM networks was taken as an optimized model. The first network encrypts 4-bit messages into 7-bit messages. The second produces the inverse transformation.

This choice of architecture has several features:

1. The authors gave an example of a working cyclic code (7, 4, 3). The possibility of solving this problem with this architecture was confirmed. Theoretically, a system of two neural networks can, after the optimization process, use a cyclic code.

2. Due to the use of LSTM nodes at the core of the architecture, the system has some internal memory. This can positively affect the quality of encoding and decoding (if there is some relationship between messages).

Neural networks were implemented on Pytorch, which allowed to significantly speed up the calculations, as well as simplify the source code.

As an optimized function, we take the standard error between the original and decoded message.

For further comparison, we calculate the upper estimate of the mean square error for the cyclic code (7, 4, 3).

To do this, suppose that when a fatal error occurs, all 4 bits are decoded incorrectly. Using (8), we obtain the following value for the expected number of errors among n bits:

$$n\,(1 + (1-p)^7 - 7p(1-p)^6), \tag{9}$$

For experimental evaluation $p = 0{,}01, n = 1000.$

Substituting the values in (9), we obtain the expected number of error bits: 2. Which leads to an estimate of the mean square error for the cyclic code (7, 4, 3): 0.002.

The plan of the experiment:

1. we will change the maximum number of iterations for the optimization process N from 100 to 300, in increments of 100;

2. we will carry out a significant number of iterations (more than 30), using each implementation to solve the problem;

3. we write out the average values of the simulation results and the optimization time.

V. Comparative analysis of the results

The data obtained as a result of the experiment are shown in table 1. In the table, the average values of the mean square errors obtained after the optimization are used as the results. In other words, the smaller the result, the better the algorithm works.

TABLE I. OPTIMIZATION VALUES FOR DIFFERENT NUMBERS OF ITERATIONS

Number of iterations, *N*	**Simple ES**		**Heuristic ES**		**CMA-ES**	
	result	***time (s)***	***result***	***time (s)***	***result***	***time (s)***
100	0.2582	101.5	0.2477	101.8	0.2334	373.6
200	0.2414	204.1	0.2303	210.7	0.2171	739.7
300	0.2485	293.7	0.2273	290.4	0.2103	1128.3

As can be seen from table 1, evolutionary strategies have solved the problem, but with a convergence rate not applicable to practical problems. The problem posed belonged to the class of instruction with a teacher. It is known that the speed of evolutionary strategies for such problems is not high [1].

There is an assumption that such a slow convergence could indicate that:

1. network architecture and / or number of nodes were not suitable for solving this problem;

2. the task was quite complex and most of the solutions were be bad, which did not allow the algorithm to find a relatively good solution in an acceptable time;

3. the task was unstable in the sense that a small change in a good solution dramatically worsens the result.

We considered each hypothesis separately.

The results of solving the problem using the Adam [9] method are shown in table 2.

As a result, the mean value of the mean square error was used. Table 2 shows that this model can solve the problem, though not as good as the cyclic code. It is worth paying attention to the fact that the error obtained using Adam is almost two orders of magnitude smaller than the error of evolutionary strategies.

TABLE II. OPTIMIZATION VALUES FOR DIFFERENT N OBTAINED USING ADAM

N	**Result**	**Time (s)**
100	0.2415	174.7
200	0.1383	334.5
300	0.1027	515.1
600	0.0094	1036.8

Consider the second hypothesis. We randomly selected 500 000 solutions. Each net weight was taken from the normal distribution with zero mean and standard deviation equal to 1.5. This covered most of the possible solutions, since optimization started from the zero point and takes no more than 300 steps of length of the order of 10^{-3}. The obtained values allowed us to calculate the mean and standard deviation of 0.3086 and 0.0207, respectively. This suggested that most of the possible solutions are unsatisfactory.

To consider the third hypothesis, we took the solution obtained at step 600 of optimization by the Adam method. Added to this solution a Gaussian noise with zero mathematical expectation and a standard deviation of 10^{-3}. This allowed us to check how sustainable the solution was. After generating 500 000 solutions from a given area, we got the following results: the expected value is 0.1622 and the standard deviation is 0.01526. This allowed us to talk about the instability of the obtained solution.

Thus, based on the data obtained, the following conclusion was made. Evolutionary strategies have an impressive list of advantages, however, if it is possible to calculate the gradient of the optimized function and the task is unstable, then it is reasonable to think about using first or second order methods.

On the other hand, evolutionary strategies make it possible to use the computing resources of an entire cluster without significant overhead, this can even out the difference in speed, and sometimes even surpass classical methods.

VI. CONCLUSION

In the course of this work, methods of evolutionary strategies and their application to the solution of the problem of error-correcting coding were considered. Several different methods and the environment for their experimental testing were implemented. Conclusions are drawn on the applicability of this approach to solving machine learning problems with a teacher.

Many experiments have been conducted. Based on the data obtained, an analysis of the effectiveness of the methods was carried out, and three hypotheses describing the results were constructed and tested.

An important result of this work is testing the applicability of evolutionary strategies for solving various problems. Moreover, the study confirmed the limitations of evolutionary strategies for the class of problems.

It is concluded that evolutionary strategies solve well reinforcement learning problems and worse solve more traditional problems. However, their disadvantages can be eliminated by connecting more computing resources.

REFERENCES

[1] S. Khaikin, “Neural networks: full course,” M.: Williams, 2008, 1104 p.

[2] H.G. Beyer and H.P. Schwefel, “Evolution strategies. A comprehensive introduction,” Natural computing, vol. 1, no. 1, pp. 3-52, 2002.

[3] T. Salimans, J. Ho, X. Chen, S. Sidor and I. Sutskever, “Evolution Strategies as a Scalable Alternative to Reinforcement Learning,” arXiv preprint arXiv:1703.03864, 2017.

[4] J. Lehman, J. Chen, J. Clune and K. O. Stanley, “ES Is More Than Just a Traditional Finite-Difference,” Proceedings of the Genetic and Evolutionary Computation Conference, pp. 450-457, 2018.

[5] E. Conti, V. Madhavan, F. Petroski Such, J. Lehman, K. O. Stanley and J. Clune, “Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-Seeking Agents,” Advances in neural information processing systems, pp. 5027-5038, 2018.

[6] A.N. Kovartsev, “A deterministic evolutionary algorithm for the global optimization of morse cluster,” Computer Optics, vol. 39, no. 2, pp. 234-240, 2015. DOI: 10.18287/0134-2452-2015-39-2-234-240.

[7] N. Hansen, “The CMA Evolution Strategy: a Tutorial,” 2009.

[8] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, “Optimization by simulated annealing,” Science, vol. 220, no. 4598, pp. 671-680, 1983.

[9] D. P. Kingma and J. Ba. Adam, “A method for stochastic optimization,” arXiv preprint arXiv:1412.6980, 2014.