The research and development of the hybrid algorithm based on the collective behavior of Fish schools and the classical optimization methods

L Demidova¹ and A Gorchakov¹,²

¹MIREA – Russian Technological University, Vernadsky pr., 78, Moscow, 119454, Russia
²E-mail: worldbeater-dev@yandex.ru

Abstract. In this paper, we consider the hybridization problem of the evolutionary-inspired Fish School Search algorithm. The algorithm shows accurate results on solving multi-extremal optimization problems, outperforms most other swarm intelligence algorithms and is computationally inexpensive. But, the accuracy of the algorithm highly depends on the defined iteration count to compute. The accuracy increases with the increase of iteration count, but this also affects computation time required to find the suboptimal solution. In this paper, we propose two hybrid approaches which intend to accelerate the Fish School Search algorithm using either gradient descent or Newton’s algorithm. We provide the comparative analysis of the original algorithm and the two hybrid algorithms applied to the Styblinsky-Tang, Eggholder and other test functions for optimization, including multi-extremal and multi-dimensional ones. The obtained results confirm that the proposed hybrid algorithms are more effective, compared to the original population-based algorithm. The results demonstrate the strong advantage of the hybrid algorithm, where the gradient descent is used as its part. Additionally, we provide the solution of the XOR problem using a perceptron with one hidden layer, where the most effective hybrid approach is used to train the network. We visualize the loss function surface near the global extreme point to show the effectiveness of the proposed hybrid algorithm.

1. Introduction

The recent research in artificial intelligence leads to intellectual systems being used almost everywhere in modern economic sectors. Decision support systems, expert systems, and forecasting systems are being used in many enterprises. This eventually leads to optimization problems occurring more often. For example, such problems can be either related to economic parameter optimization, or to the determination of chemical composition of alloys [1]. Mathematical optimization is the selection of the best element from some set of available alternatives. In the simplest case, an optimization problem consists of maximizing or minimizing a real function \( f: X \to \mathbb{R} \), where \( \mathbb{R} \) is the set of real numbers, by systematically choosing \( \mathbf{x} \in X \) and computing the values of \( f(\mathbf{x}) \). When minimizing the function, we search for such \( \mathbf{x}_0 \), that \( \forall \mathbf{x} \in X: f(\mathbf{x}_0) \leq f(\mathbf{x}) \). When maximizing the function, we search for such \( \mathbf{x}_0 \), that \( \forall \mathbf{x} \in X: f(\mathbf{x}_0) \geq f(\mathbf{x}) \).

In order to solve optimization problems, one can use evolutionary-inspired (population-based) algorithms. Population-based algorithms process a variety of solutions to an optimization problem at a
time. The population consists of agents, and on every iteration, each of the agents affects the population behavior as the whole. There are many kinds of population-based algorithms, including genetic algorithms [2], swarm intelligence algorithms [3], ant colony algorithm [4], bee swarm algorithm [5], fish school search algorithm and others. Modifications of these algorithms exist, applied to solve practical problems. Such problems include classifier and regression model parameter optimization based on SVM algorithms [6] or random forests [7], architecture optimization of neural networks [8]. In this paper, we consider the fish school search algorithm and its hybrid versions, based on the combination of the evolutionary-inspired algorithm with such classical optimization methods, as the gradient descent and Newton’s algorithm in optimization.

2. The Fish School Search Algorithm

The population-based Fish School Search algorithm (FSS) was proposed by Bastos Filho and Lima Neto in 2008 [9]. The aims of the proposed approach include the minimization of time required to find the optimal solution and the elimination of the premature convergence problem, which is inherent to many population-based optimization algorithms. Premature convergence means the convergence of an algorithm before the global optimum solution is reached.

In Fish School Search, a fish school is a set of agents moving at a similar speed in the same direction. When solving the maximization problem, the core idea of the algorithm is to make the agents move towards the positive gradient of the function in order to eat and gain weight. The agents with larger weight have a greater impact on the population as a whole. Each iteration of the algorithm includes three movement operators and one feeding operator. On every iteration the best agent \( \tilde{x}_{\text{best}} \) is chosen.

The first operator is the individual movement operator, which can be described as: \( \tilde{x}_{i,t+1} = \tilde{x}_{i,t} + \text{step}_{\text{ind}} \hat{r} \), where \( \hat{r} \) is the vector containing random real numbers which are uniformly distributed on \([-1, 1]\). \( \text{step}_{\text{ind}} \) is the maximum displacement, \( \tilde{x}_{i,t} \) is the position of the \( i \)-th agent on the \( t \)-th iteration. The new position is only accepted, when \( f(\tilde{x}_{i,t+1}) > f(\tilde{x}_{i,t}) \). For the three-dimensional Cartesian coordinate system, where the third coordinate is the fitness function value, the individual movement operator can be described as:

\[
\begin{align*}
\left( x_{i,t+1}, y_{i,t+1}, z_{i,t+1} \right) &= \left( x_{i,t}, y_{i,t}, z_{i,t} \right) + \text{step}_{\text{ind},t} \left( \text{rand}[\{-1, 1\}] \right). 
\end{align*}
\] (1)

For each agent \( \tilde{x}_{i,t} \) the \( \Delta f_{i,t+1} = f(\tilde{x}_{i,t+1}) - f(\tilde{x}_{i,t}) \) value is computed. Each agent \( \tilde{x}_{i,t} \) has its own scalar weight value \( w_{i,t} \). After the individual movement operator is applied, the weight of each \( \tilde{x}_{i,t} \) agent is updated, according to the following formula:

\[
\begin{align*}
w_{i,t+1} &= w_{i,t} + \frac{\Delta f_{i,t+1}}{\max(\{|\Delta f_{i,t+1}|\})}.
\end{align*}
\] (2)

Then, the collective-instantaneous movement operator gets applied. As a result, the agents with the best positions attract all other agents of the population. For each agent the \( \Delta \tilde{x}_{i,t+1} = \tilde{x}_{i,t+1} - \tilde{x}_{i,t} \) value gets computed. The collective-instantaneous movement occurs according to \( \tilde{x}_{i,t+1} = \tilde{x}_{i,t} + \vec{t}_{t+1} \), where \( \vec{t}_{t+1} \) is the vector, computed as such:

\[
\vec{t}_{t+1} = \frac{\sum_{i=1}^{N} \Delta \tilde{x}_{i,t+1} \Delta f_{i,t+1}}{\sum_{i=1}^{N} \Delta f_{i,t+1}}.
\] (3)

Next, the barycenter vector \( \overrightarrow{B}_{t+1} \) is computed as:

\[
\overrightarrow{B}_{t+1} = \frac{\sum_{i=1}^{N} \tilde{x}_{i,t+1} \Delta w_{i,t+1}}{\sum_{i=1}^{N} \Delta w_{i,t+1}}.
\] (4)
Finally, the collective-volatile movement operator gets applied according to:

\[
\vec{x}_{i,t+1} = \vec{x}_{i,t+1} + step_{vol,t} \vec{r} \cdot \frac{\vec{x}_{i,t+1} - \vec{B}_{t+1}}{||\vec{x}_{i,t+1} - \vec{B}_{t+1}||}.
\]  

(5)

Here, \(\vec{r}\) is the vector containing random values, uniformly distributed on \([-1,1]\), \(\vec{B}_{t+1}\) is the barycenter vector. In case if the total weight of the population has increased since the last iteration, the agents are attracted to the barycenter (the ‘−’ sign is used in the formula (5)), otherwise the agents are spread away from the barycenter (the ‘+’ sign is used in the formula (5)).

The variables \(step_{ind}\) and \(step_{vol}\) decay linearly on each iteration. In case if we are solving the maximization problem, in the end of each iteration we find such \(\vec{x}_{t,\text{best}}\), that \(\forall \vec{x}_t \in \mathbb{P}_t : f(\vec{x}_{t,\text{best}}) \geq f(\vec{x}_t)\), where \(\mathbb{P}_t\) is the population on \(t\)-th iteration. The algorithm stops, when the maximum iteration count \(iter_{\text{max}}\) is reached. The \(\vec{x}_{\text{best}}\) position, computed on the final iteration, is considered to be the solution.

3. Implementation and approbation of the Fish School Search Algorithm

The optimization algorithm inspired on the collective behavior of fish schools was implemented using Python – the interpreted, high-level, general-purpose programming language. The NumPy and SciPy libraries were used, as well as the freely distributed Jupiter Notebook environment with the IPython shell. To test the Fish School Search algorithm on maximizing the multi-extremal real function, the Styblinsky-Tang function \([10]\) was chosen, multiplied by \(-1\):

\[
f(\vec{x}) = \frac{\sum_{i=1}^{n} x_i^4 - 16x_i^2 + 5x_i}{-2}, x_i \in [-5,5], i \in [1,n].
\]  

(6)

In the three-dimensional Cartesian coordinate system, the Styblinsky-Tang function (6) is given by:

\[
f(x, y) = -0.5(x^4 - 16x^2 + 5x) - 0.5(y^4 - 16y^2 + 5y).
\]  

(7)

It is a known fact, that the function (7) has its global maximum at \(P_{\text{max}}(-2.903534,-2.903534)\), and \(78.33232 < f(P_{\text{max}}) < 78.33234\). The programmatic optimization of the Styblinsky-Tang function was performed under the following conditions: the search area was defined as \(x \in [-5,5], y \in [-5,5]\), the fish school search population consisted of 50 agents, \(iter_{\text{max}} = 25\). The maximum agent displacements on the individual and collective-volatile movement steps were defined as \(step_{ind} = 2.5, step_{vol} = 5\). The population of agents was visualized at the 0-th, 5-th and the 25-th iteration (figure 1).

![Figure 1](image-url)  

**Figure 1.** Visualization of the Fish School Search algorithm population of the 0-th, 5-th and 25-th iterations for the Styblinsky-Tang function multiplied by \(\epsilon=1\).

As a result, we obtained the following suboptimal solution: \(\vec{x}_{\text{best}} = (-2.90539426, -2.9177174)\). Increasing the iteration count \(iter_{\text{max}}\) further didn’t...
show significant improvement. In the best case, the error was in the third digit after decimal point. The population-based algorithm successfully optimized the multi-extremal function. But, to increase the accuracy of the obtained results, it is reasonable to use the classical optimization algorithms, starting the optimization process from the best solution \( \hat{x}_{best} \) found with the evolutionary-inspired algorithm.

4. The Hybrid Algorithm based on the Fish School Search and the Newton’s method

The Newton’s method is an iterative method for finding roots of a differentiable function \( f \). The method solves the \( f(x) = 0 \) equation. In optimization, the Newton’s method is applied to the twice-differentiable function \( f \), to find the solutions of the \( f'(x) = 0 \). The solutions are the stationary points of \( f \), and can either be minimum, maximum or saddle points [11]. In the multidimensional space, the iterative Newton’s optimization scheme is given by:

\[
\hat{x}_{n+1} = \hat{x}_n - [Hf(\hat{x}_n)]^{-1} \nabla f(\hat{x}_n).
\]

Here, \([Hf(\hat{x}_n)]^{-1}\) is the inverse of the Hessian matrix, \( \nabla f(\hat{x}_n) \) is the gradient of the function \( f \) at the point \( \hat{x}_n \), \( n \) is the iteration number. In the three dimensional Cartesian coordinate system, where the third coordinate is the fitness function value, the Newton’s iterative scheme can be described as:

\[
\begin{pmatrix}
\hat{x}_{n+1} \\
\hat{y}_{n+1}
\end{pmatrix} = \begin{pmatrix}
\hat{x}_n \\
\hat{y}_n
\end{pmatrix} - \begin{pmatrix}
\frac{\partial^2 f}{\partial x^2} (\hat{x}_n, \hat{y}_n) & \frac{\partial^2 f}{\partial x \partial y} (\hat{x}_n, \hat{y}_n) \\
\frac{\partial^2 f}{\partial y \partial x} (\hat{x}_n, \hat{y}_n) & \frac{\partial^2 f}{\partial y^2} (\hat{x}_n, \hat{y}_n)
\end{pmatrix}^{-1} \begin{pmatrix}
\frac{\partial f}{\partial x} (\hat{x}_n, \hat{y}_n) \\
\frac{\partial f}{\partial y} (\hat{x}_n, \hat{y}_n)
\end{pmatrix}
\]

Partial derivatives of the given function \( f \), including second-order partial derivatives, can be obtained using finite difference approximation [12]. The iterative optimization process continues until the stop condition \(|x_{i,n+1} - x_{i,n}| \leq \varepsilon\) is not achieved, where \( \varepsilon \) is the allowed error and \( x_{i,n} \) is the \( i \)-th component of the \( \hat{x}_n \) vector at \( n \)-th iteration.

In order to improve the Fish School Search algorithm accuracy, we run \( \text{iter}_{max} \) iterations of the population-based algorithm. The algorithm finds the best solution \( \hat{x}_{best} \). Then, the \( \hat{x}_{best} \) vector is used as the starting point for Newton’s optimization algorithm, so Newton’s method converges to the closest to \( \hat{x}_{best} \) stationary point. Notably, if there is a saddle point or a minimum point near the \( \hat{x}_{best} \) vector, it is possible that Newton’s optimization algorithm won’t converge to the maximum point.

5. The Hybrid Algorithm based on the Fish School Search and the gradient descent

Gradient descent is an iterative optimization algorithm for finding the minimum of a function. To find the local minimum of a function \( f \), one takes steps proportional to the negative of the gradient \( \nabla f \) of the function \( f \) at the given point \( \hat{x} \). To find the local maximum of a function \( f \), one takes steps proportional to the positive of the gradient \( \nabla f \), this algorithm is also known as the gradient ascent. When minimizing the function \( f \), the gradient decent formula is given by:

\[
\hat{x}_{n+1} = \hat{x}_n - \gamma \nabla f(\hat{x}_n).
\]

When maximizing the function \( f \), the iterative gradient ascent formula is given by:

\[
\hat{x}_{n+1} = \hat{x}_n + \gamma \nabla f(\hat{x}_n).
\]

Here, \( \hat{x}_n \) is the point at the \( n \)-th iteration, and \( \gamma \) is the learning rate. The convergence to the local extreme point is guaranteed, when the \( \gamma \) value gets calculated on each iteration according to:

\[
\gamma_n = \frac{|(\hat{x}_n - \hat{x}_{n-1})^T (\nabla f(\hat{x}_n) - \nabla f(\hat{x}_{n-1}))|}{||\nabla f(\hat{x}_n) - \nabla f(\hat{x}_{n-1})||^2}.
\]

To improve the characteristics of the Fish School Search algorithm, we run \( \text{iter}_{max} \) iterations of the population-based algorithm. Then, we use the best solution \( \hat{x}_{best} \) as the starting point for the gradient descent (10) (or for the gradient ascent (11), in case if we are maximizing the function).
6. Optimizing the Styblinsky-Tang Function using the original and the hybrid algorithms

To compare the accuracy and performance of the original population-based algorithm and the hybrid algorithms, the Styblinsky-Tang test function (7) was chosen. This function was maximized in the three-dimensional Cartesian coordinate system using the original algorithm, the population-based algorithm combined with Newton’s method, the population-based algorithm combined with the gradient descent.

The Styblinsky-Tang function was optimized under the following conditions: for the population-based algorithm the search boundaries were defined as $x \in [-5, 5]$, $y \in [-5, 5]$, 100 agents were spawned, step sizes were set according to $step_{ind} = 2.5$, $step_{pol} = 5$. We repeated the experiments twice, first time for $iter_{max} = 50$, second time for $iter_{max} = 100$. For the gradient descent we used $\gamma = 0.25$. For both the Newton’s optimization algorithm and for the gradient descent algorithm we used $\varepsilon = 10^{-6}$. Each of the algorithms was evaluated 10 times. In each case, the same pseudo random number generator seed [13] was used. The plots we obtained are shown in figure 2, the accuracy comparison is shown in table 1.

![Figure 2. Convergence of the proposed hybrid optimization algorithms, $iter_{max} = 50$, $\varepsilon = 10^{-6}$.](image)

| Algorithm                      | Mean      | Variation       | Std. Dev.       |
|-------------------------------|-----------|-----------------|-----------------|
| Fish School Search, 50 iterations | 77.892    | 1.294e-01       | 3.598e-01       |
| Fish School Search, 100 iterations | 78.229    | 7.037e-03       | 8.389e-02       |
| FSS + Newton’s alg., 50 iterations | 78.332    | 1.212e-28       | 1.101e-14       |
| FSS + Newton’s alg., 100 iterations | 78.332    | 4.093e-29       | 6.355e-15       |
| FSS + Gradient ascent, 50 iterations | 78.332    | 1.009e-28       | 1.005e-14       |
| FSS + Gradient ascent, 100 iterations | 78.332    | 6.058e-29       | 7.784e-15       |

The time required for each of the algorithms to converge was measured as well. To handle the experiments, the computer running under Ubuntu 18.04 LTS was used, with Intel® Core™ i7-4770 CPU (3.40 GHz clock frequency), with 16GB of RAM. The algorithms, as already mentioned, were implemented in Python and tested in the IPython shell with the Jupiter GUI. Each of the algorithms was evaluated 10 times. We computed characteristics of the test run, such as the mean value, the variation, the standard derivation. The results are presented in table 2.

| Algorithm                      | Mean (s)  | Variation (s) | Std. Dev. (s) |
|-------------------------------|-----------|---------------|---------------|
| Fish School Search, 50 iterations | 0.699     | 0.035e-03     | 0.595e-02     |
| Fish School Search, 100 iterations | 1.390     | 0.118e-03     | 1.086e-02     |
| FSS + Newton’s alg., 50 iterations | 0.749     | 0.359e-03     | 1.895e-02     |
7. Optimizing the Eggholder Function using the original and the hybrid algorithms

The next candidate for optimization is the Eggholder function \([14]\), multiplied by \(-1\). In the three-dimensional Cartesian coordinate system, when \(x \in [-512, 512], y \in [-512, 512]\), the function is given by (13), the plot of the function is show in figure 3.

\[
f(x, y) = (y + 47)\sin\sqrt{\frac{x^2}{2} + (y + 47)} + x\sin\sqrt{|x - (y + 47)|}. \tag{13}
\]

![Figure 3. The plot of the Eggholder function (10), \(x \in [-512, 512], y \in [-512, 512]\).](image)

It is known, that the Eggholder function has its global maximum at \(P_{\text{max}}(512, 404.2319)\), where \(f(P_{\text{max}}) \approx 959.6407\). The function was maximized using the three algorithms, including the original Fish School Search algorithm and the two hybrids. The programmatic optimization of the Eggholder function was made under the following conditions: the search boundaries were defined as \(x \in [-512, 512], y \in [-512, 512]\), 512 agents were spawned, and initial step sizes were defined as \(\text{step}_{\text{ind}} = 512, \text{step}_{\text{vol}} = 1024\). Again, we repeated the experiments twice, for \(\text{iter}_{\text{max}} = 50\), and for \(\text{iter}_{\text{max}} = 100\). For the gradient ascent we used \(\gamma = 0.5\) initial step size, for both the gradient ascent and for Newton’s optimization algorithm we used \(\varepsilon = 10^{-6}\). The plots showing the convergence of the hybrid algorithms are presented in figure 4.

![Figure 4. The convergence of the hybrid algorithms when \(\text{iter}_{\text{max}} = 50, \varepsilon = 10^{-6}\).](image)

| Algorithm | Mean \(x\) | Variance \(x\) | Std. Dev. \(x\) |
|-----------|------------|----------------|-----------------|
| Fish School Search, 50 iteration | 952.838 | 1.094e+02 | 1.046e+01 |
| Fish School Search, 100 iterations | 956.258 | 2.009e+01 | 4.482e+00 |
| FSS + Newton’s alg., 50 iterations | 107.234 | 4.523e+05 | 6.725e+02 |
| FSS + Newton’s alg., 100 iterations | 56.439 | 2.929e+05 | 5.413e+02 |
| FSS + Gradient ascent, 50 iterations | 959.641 | 3.877e-26 | 1.969e-13 |
| FSS + Gradient ascent, 100 iterations | 959.641 | 2.585e-26 | 1.608e-13 |
Table 4. Time, required for the algorithms to converge when optimizing the Eggholder function.

| Algorithm                        | Mean (s) | Variation (s) | Std. Dev. (s) |
|----------------------------------|----------|---------------|---------------|
| Fish School Search, 50 iterations| 5.072    | 0.001         | 0.035         |
| Fish School Search, 100 iterations| 10.141  | 0.003         | 0.050         |
| FSS + Newton’s alg., 50 iterations| 5.442   | 0.179         | 0.424         |
| FSS + Newton’s alg., 100 iterations| 10.768  | 0.036         | 0.189         |
| FSS + Gradient ascent, 50 iterations| 5.073   | 0.001         | 0.021         |
| FSS + Gradient ascent, 100 iterations| 10.172  | 0.002         | 0.043         |

In figure 4 we see, that the hybrid optimization algorithm which uses Newton’s optimization method didn’t converge to the global maximum. To measure time and accuracy, each of the algorithms was run on the test device 10 times, the mean value, the variation and the standard deviation parameters were computed. The results are shown in tables 3 and 4.

Additionally, the hybrid algorithms were benchmarked on such two-dimensional test functions for optimization, as the Booth function [15], the Matyas function [16]; on such multidimensional test functions, as the multi-extremal Rastrigin function [17], the Rosenbrock function [18]; on other test functions for optimization, commonly used to benchmark optimization algorithms [19] (20 total). We also considered the multi-extremal version of the Styblinsky-Tang function (6). For the multidimensional functions, we used from 3 to 10 variables. In each of the experiments the algorithms show similar results, from which we can assume, that combining the population-based Fish School Search algorithm with such classical optimization algorithms, as Newton’s method and the gradient descent (or, with the gradient ascent), can improve accuracy of the algorithm, at the cost of minor time losses. The use of the hybrid algorithm, based on Newton’s optimization method, implies a number of limitations – in some cases, when one optimizes a multi-extremal function, the method can converge to the wrong solution, for example, to a saddle point. Another hybrid algorithm, based on the gradient descent, can be used in such cases when the function $f$ is differentiable in the study region. In this case, the Fish School Search algorithm chooses the approximated suboptimal solution $\hat{x}_{\text{best}}$, and the gradient descent algorithm helps to improve the accuracy of the estimated solution.

8. Solving the XOR problem

The proposed hybrid optimization algorithm, based on the Fish School Search algorithm and the gradient descent, can be used to train neural network models. A neural network is a programmatic system, which allows making decisions using the evolution of a complex non-linear system. A neural network takes in a vector consisting of input data, and the output signal encodes the decision made by the system [20]. Perceptron is a mathematical model, proposed by Frank Rosenblatt in 1957. Perceptron was one of the first neural network models. Perceptrons allow creating a set of associations between the input data and the desired reaction on the network output.

To try the proposed hybrid optimization algorithm we’ve chosen the linearly inseparable XOR problem. This task is curious, because the Boolean XOR function is linearly inseparable, compared to the AND function and the OR function [21]. Hence, the XOR function is often used to benchmark optimization algorithms [22]. The XOR function truth table is shown in table 5.

Table 5. The XOR function.

| X1 | X2 | Y   |
|----|----|-----|
| 0  | 0  | 0   |
| 0  | 1  | 1   |
| 1  | 0  | 1   |
| 1  | 1  | 0   |

When solving the XOR problem using a perceptron, the input data is as a set of $X$ objects, and the output data is a set of $Y$ answers. Here, the variables and the answers belong to the set $\{0, 1\}$. The goal
is to restore the function $f^* : \mathbb{X} \to \mathbb{Y}$. To solve the described problem, we built a perceptron with one hidden layer, which contains 8 neurons. The perceptron model is shown in figure 5.

Other neural network architectures can be used to solve the XOR problem [23], more efficient perceptron architectures, including those that contain more hidden layers, different number of neurons, or use polynomial transformations [24]. But to try the hybrid algorithm we decided to use the described architecture to increase the number of conflicting possible solutions. In our perceptron, we use the hyperbolic tangent activation function on the hidden layer, which is given by:

$$th(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$ \hspace{1cm} (14)

Here, $x$ is the input signal multiplied by neuron’s weights. For the output layer, we use the sigmoid activation function. The sigmoid activation function is given by:

$$\sigma(x) = \frac{1}{1 + e^x}$$ \hspace{1cm} (15)

In the XOR problem we consider, the input objects from the $\mathbb{X}$ set should be assigned either to class 0 or class 1, so we use here the binary cross-entropy activation function, given by:

$$L(\mathbb{X}, \mathbb{Y}, \mathbf{w}) = -\frac{1}{N} \sum_{i=0}^{N} (y_i \log f(\tilde{x}_i, \mathbf{w}) + (1 - y_i) \log (1 - f(\tilde{x}_i, \mathbf{w}))).$$ \hspace{1cm} (16)

Here, $y_i$ is the class label, $y_i \in \mathbb{Y}$, $f(\tilde{x}_i, \mathbf{w})$ is the predicted probability that the point $\tilde{x}_i$ should be assigned to the $y_i$ class, $\tilde{x}_i \in \mathbb{X}$, $N$ denotes the elements count in the training set $\mathbb{X}$, $\mathbf{w}$ is the weights.

![Figure 5. Perceptron built for solving the XOR problem.](image)

Notably, the set of objects $\mathbb{X}$ and the set of answers $\mathbb{Y}$ don’t change during the perceptron training process, the loss function depends only on the weights $\mathbf{w}$ of the neural network. This means that the process of training the perceptron leads to minimizing the loss function with respect to the weights $\mathbf{w}$. 
Neural networks contain many parameters, and so the loss functions live in a very high-dimensional space, but visualizations are only possible using line or surface plots. Several approaches exist which allow visualizing loss functions. In one of the proposed approaches [25], one chooses a central point $\bar{w}$ and two orthogonal direction vectors, $\vec{\delta}$, and $\vec{\eta}$. Then, the multi-dimensional loss function gets transformed into:

$$f(\alpha, \beta) = L(\mathcal{X}, \mathcal{Y}, \bar{w} + \alpha \vec{\delta} + \beta \vec{\eta}).$$  

(17)

Here, $\vec{\delta}$ and $\vec{\eta}$ are the chosen orthogonal vectors, $L$ is the multi-dimensional loss function of a neural network, $\bar{w}$ is the suboptimal vector containing the weights of a neural network, $\mathcal{X}$ is the set of input objects, $\mathcal{Y}$ is the set of answers, $\alpha$ and $\beta$ are the scalar coefficients. The loss function of the perceptron we built to solve the XOR problem takes in 33 parameters, which are the weights of our perceptron.

Using the interpreted, general-purpose Python language, and the Keras library, we implemented the perceptron described above. The loss function was optimized twice, using different sets of the hybrid algorithm parameters.

In the first case, we used the following parameters for the Fish School Search part of the hybrid algorithm: $r = 10$, $\text{step}_{\text{ind}} = 5$, $\text{step}_{\text{vol}} = 10$, $\text{iter}_{\text{max}} = 15$. 50 agents were spawned. Here, the $r$ variable scalar value denotes, that the weights, which are randomly initialized by the Keras library, are defined on the interval $[w_{i,\text{initial}} - r, w_{i,\text{initial}} + r]$. For the gradient descent part of the hybrid algorithm, we used initial step size $\gamma = 0.5$, $\varepsilon = 10^{-4}$. The landscape of the multidimensional loss function was visualized according to (17), when $\alpha \in [-10,10]$, $\beta \in [-10,10]$. The results are shown in figure 6. In the second case, we choose the following parameters for the Fish School Search part of the hybrid algorithm: $r = 4$, $\text{step}_{\text{ind}} = 4$, $\text{step}_{\text{vol}} = 4$, $\text{iter}_{\text{max}} = 15$. Again, 50 agents were spawned. For the gradient descent we choose the step size $\gamma = 0.5$ and $\varepsilon = 10^{-4}$. To visualize the landscape of the multidimensional loss function according to (17), we assumed $\alpha \in [-4,4]$, $\beta \in [-4,4]$. The results are shown in figure 7.

**Figure 6.** Visual representation of the results: a – Hybrid algorithm convergence when $r = 10$; b – Loss function surface visualization near the suboptimal solution; c – Contour plot of the loss function near the suboptimal solution.

During the solution of the XOR problem with the perceptron and the hybrid optimization algorithm, while using $r = 10$ we got the solution after 16 iterations. When using $r = 4$, it took 26 iterations for the hybrid optimization algorithm to converge. The suboptimal perceptron weights $\bar{w}_1$ and $\bar{w}_2$ we got in the first and in the second cases, differ. We observed, that in the first case $\forall w_i \in \bar{w}_1$, $|w_i| \leq 10$, and in the second case $\forall w_i \in \bar{w}_2$, $|w_i| \leq 4$. When using $r = 10$, the error in the predicted XOR function values locates in the second digit after decimal point, and when using $r = 3$ – in the first digit after decimal point. Assuming the fact, that in both experiments we used the same pseudorandom number generator seed value, we suppose, that the obtained solutions $\bar{w}_1$ and $\bar{w}_2$ can be
suboptimal only inside the given boundaries, limited by the \( r \) parameter. If we increase the \( r \) parameter value, we can get other, probably better, solutions.

![Figure 7](image)

**Figure 7.** Visual representation of the results: a – Hybrid algorithm convergence when \( r = 4 \); b – Loss function surface visualization near the suboptimal solution; c – Contour plot of the loss function near the suboptimal solution.

If we analyze the surfaces of the loss functions, obtained using the (17) formula, we notice a number of local minima points, which can prevent the classical optimization algorithms from finding the most optimal solutions. The use of the hybrid algorithm based on the Fish School Search allows us to choose the best promising region of many, and the gradient descent approximates the solution to the closest local minima with better accuracy.

9. **Conclusion**

When benchmarking the hybrid optimization algorithms on the test functions for optimization, including the Eggholder, Booth, Styblinsky-Tang, Rastrigin, Rosenbrock functions and others, including multi-extremal versions of some of the test functions for optimization, the hybrid algorithm based on the combination of the evolutionary-inspired Fish School Search and the gradient descent with the variable step size, showed the best results. The described hybrid algorithm can be used as a learning algorithm for a perceptron or for a more complex neural network, as well as for hyperparameters tuning of neural networks. During the research, the hybrid algorithms were implemented using the Python programming language. In the future, we are planning to implement the optimizer using Keras and Tensorflow declarative programmatic APIs, to achieve acceptable training speed when solving more complex tasks.

**Reference**

[1] Korneev A and Sukhanov A 2018 Investigation of accuracy and speed of convergence of algorithms of stochastic optimization of functions on a multidimensional space *Vestnik of Astrakhan State Technical University. Series: management, computer science and informatics* **3** 26–37

[2] Goodman E 2007 Introduction to genetic algorithms *Genetic and Evolutionary Computation Conf., GECCO 2007, Proc.* pp 3205–24

[3] Shi Yu and Eberhart R 1998 A Modified Particle Swarm Optimizer *Proc. 1998 IEEE Int. Conf. on Evolutionary Computation* 69–73

[4] Dorigo M, Birattari M and Stützle T 2006 Ant Colony Optimization *IEEE Computational Intelligence Magazine* **1** 28–39

[5] Reza A, Moehmmani S and Ziarati K 2010 A novel bee swarm optimization algorithm for numerical function optimization *Communications in Nonlinear Science and Numerical Simulation* **15** 3142–55

[6] Demidova L, Nikulchev E and Sokolova Yu 2016 Big Data Classification Using the SVM Classifiers with the Modified Particle Swarm Optimization and the SVM Ensembles *Int. Journal of Advanced Computer Science and Applications* **7** (5) 294–312
[7] Shah S S and Pradhan M 2016 R-GA: An Efficient Method for Predictive Modelling of Medical Data Using a Combined Approach of Random Forests and Genetic Algorithm ICTACT Journal on Soft Computing 6 1153–56

[8] Janati I, Mohammed A, Ramchoun H, Ghanou Y and Ettaouil M 2016 Genetic algorithm for neural network architecture optimization 2016 3rd Int. Conf. on Logistics Operations Management (GOL) 1–4

[9] Bastos Filho C, Lima Neto F, Lins A, Nascimento A and Lima M 2008 A novel search algorithm based on fish school behavior 2008 IEEE Int. Conf. on Systems, Man and Cybernetics, Singapore 2646–51

[10] Mesa E, Velásquez J and Jaramillo P 2015 Evaluation and Implementation of Heuristic Algorithms for Non-Restricted Global Optimization IEEE Latin America Transactions 13 (5) 1542–9

[11] Polyak B 2007 Newton’s method and its use in optimization European Journal of Operational Research 181 1086–96

[12] McClarren R 2018 Finite Difference Derivative Approximations Computational Nuclear Engineering and Radiological Science Using Python (London: Academic Press) pp 251–266

[13] L’Ecuyer P 2012 Random number generation Handbook of Computational Statistics: Concepts and Methods ed Gentle J E et al (Berlin: Springer) pp 35–71

[14] Odili J and Noraziah A 2018 African Buffalo Optimization for Global Optimization Current Science 114 627–36

[15] Balasub pareddy M 2016 Multi-objective optimization in the presence of ramp-rate limits using non-dominated sorting hybrid fruit fly algorithm Ain Shams Engineering Journal 7 895–905

[16] Bodenstein C and Zimmermann A 2015 Extending design space optimization heuristics for use with stochastic colored Petri nets 9th Annual IEEE Int. Systems Conf., SysCon 2015 Proc. pp 139–44

[17] Mühlenbein H, Schomisch D and Born J 1991 The Parallel Genetic Algorithm as Function Optimizer Parallel Computing 17 619–32

[18] Rosenbrock H 1960 An automatic method for finding the greatest or least value of a function The Computer Journal 3 (3) 175–84

[19] Hussain K, Salleh M, Cheng S and Naseem R 2017 Common Benchmark Functions for Metaheuristic Evaluation: A Review Int. Journal on Informatics Visualization 1 218–23

[20] Sigov A, Andrianova E, Zhukov D, Zykov S and Tarasov I 2019 Quantum informatics: Overview of the main achievements. Rossiyskiy tekhnologicheskij zhurnal (Russian Technological Journal) 7 (1) 5–37

[21] Gruzling N 2001 Linear Separability of the Vertices Of An n-Dimensional Hypercube Master’s Thesis, University of Northern British Columbia p 3

[22] Subadra M, Lakshmi K, Sundar J and MathiVathani K 2014 Design and Implementation of Multilayer Perceptron with On-chip Learning in Virtex-E AASRI Procedia 6 82–8

[23] Reljan-Delaney M and Wall J 2017 Solving the linearly inseparable XOR problem with spiking neural networks 2017 SAI Computing Conf. 701–5

[24] Piazza F, Uncini A and Zenobi M 1992 Artificial neural networks with adaptive polynomial activation function Int. Joint Conf. on Neural Networks 2 343–8

[25] Goodfellow I, Vinyals O and Saxe A 2015 Qualitatively characterizing neural network optimization problems 3rd Int. Conf. on Learning Representations, ICLR 2015 (arXiv:1412.6544v6)