Hybrid Zeroth- and First-Order Strategies for Training Neural Networks

Chuheng Zhang
School of Physics, Nanjing University, Nanjing 210093, China
Email: zhangchuheng123@live.com

***Abstract*—Artificial neural networks are nonlinear models popularly used in machine learning and adaptive control. Due to their nonlinearly, training a neural network with one or more hidden layers commonly requires to solve an optimization problem with many local optima. It is believed that gradient-based (first-order) optimization methods are efficient but are easily get trapped in local minima, while zeroth-order methods (e.g., evolutionary algorithms) are able to deal with local minima but are regarded to have low efficiency. Is it possible to combine the two kinds of methods to inherit both of their advantages? This paper designs several hybrid strategies combining the back-propagation method, a typical gradient-based training method, with the RACOS method, a recently proposed derivative-free optimization method with theoretical support. We compare the strategies on training a multilayer feedforward neural network on MNIST dataset. Experiments show that one strategy is able to achieve better optimization performance in both quality and efficiency.**

***Index Terms*—multilayer feedforward neural network, hybrid strategy, RACOS, back-propagation**

I. Introduction

Artificial neural networks [2] are popular models in machine learning and adaptive control. They are very flexible to represent complex functions, and thus the training of a large neural network is not a trivial task. The training of multilayer feedforward networks is to minimize some loss function by adjusting the weights connecting neurons. Due to the nonlinearity in neural networks, the landscape of the optimization is usually very complex, e.g., has a lot of local minima.

Feedforward networks for supervised learning are canonically trained by back-propagation methods [3], which employ the gradient decent method to minimize the loss function incorporating the chain-rule of the gradient. Gradient methods (also called first-order methods) are usually considered efficient for convex functions. However, for feedforward networks with hidden layers, the overall optimization function is hard to be convex. Therefore, back-propagation methods can be trapped in local minima, or even converge to only stationary points. Although many variants of gradient or Newton-like methods (second order methods) have been studied [2], optimizing such networks is still considered hard.

Meanwhile, zeroth-order methods, such as evolutionary algorithms [1], are better suitable for optimizing complex functions. Instead of using gradient, zeroth-order methods employ heuristic sampling strategies to sample solutions, thus can avoid the miss-leading of the gradient. However, also because the missing of the gradient information, zeroth-order methods usually converge slower than first-order methods.

We are interested in the question that can we have a combination of the zeroth- and first-order methods, so that it can inherit the advantages of the both. In this paper we investigate several strategies to combine the two methods: the first-order method is used as (a) the inner evaluator, (b) the inner optimizer, and (c) the post-optimizer for the zeroth-order method. Employing the classical back-propagation method as the representative first-order method and RACOS [4], a recently developed theoretical grounded method, as the representative zeroth-order method, we conduct experiments on optimizing a five-layer network using the MNIST hand-writing digital recognition data set. Experiment results show that using the back-propagation as the inner optimizer and poster optimizer for RACOS leads to the best optimization performance.

The rest of this paper is organized as follows. Section II, III, and IV respectively introduces the neural networks, the back-propagation method, and the RACOS method. Section V presents several hybrid strategies combining RACOS and back-propagation method for training the network. Section VI presents the experimental results. Section VII concludes this paper.

II. Artificial Neural Networks

Artificial neural networks are a family of model inspired by the biological neural networks including brains or neural tissue of animals. They have shown great potential in machine learning and adaptive control. Multilayer feedforward network that we used here for training is one of the most typical artificial neural networks and shows great potential in function approximation, classification, data processing, robotics and industrial control.

The structure of feedforward network is shown in Figure 1. It has M layers of neurons - for the network shown in the figure $M = 3$. It is composed of one input layer, one output layer and several hidden layers. There are different number of neurons in each layer and each neuron has connection to every neuron in adjacent layers. The excitation of the signal is passed from the input layer, via complex hidden layers and then to the output layer.

For each neuron, the output signal generates as follows. As is shown in Figure 2, income signals from different neurons in the former layer will be multiplied by corresponding weights,

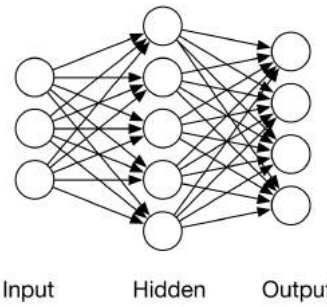

Fig. 1. Structure of multiple feedforward network. The circles represent for the neurons of the network. The arrows show the connection between the neurons in adjacent layers.

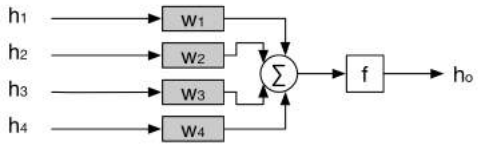

Fig. 2. Structure of a single neuron.

summed up and processed though a non-linear activation mapping. The outcome signal again serves as the input signal for neurons of the next layer or as the output signal of the entire neural network. Mathematically, it can be written as

$$h_o = f(w_1h_1 + w_2h_2 + w_3h_3 + \cdots) = f(\sum_i w_ih_i) \quad (1)$$

where $h_1, h_2, \cdots$ are the input signals from former layer, $w_1, w_2, \cdots$ are corresponding weights, h_o is the output signal and $f(\cdot)$ is the non-linear activation function.

By adjusting the weights of the network, the network will give corresponding response to given input, and thus can be used for classification or function approximation. Given a large number of sample inputs and target outputs, the weights of the network can be adjusted to response correctly to potential inputs. This process is the training of multilayer feedforward network for supervised learning. The sample inputs and the target outputs is called training data and denoted as $\{(\underline{\boldsymbol{x}}, \underline{\boldsymbol{y}})\}$, where $\underline{\boldsymbol{x}}$ and $\underline{\boldsymbol{y}}$ are sample input and corresponding target output respectively. They are both column vectors.

For the ease of notation, we rewrite the signal of the neurons on each layer and the weights between two layer in the form of vector and matrix respectively. Hence, the network structure can be written as

$$\boldsymbol{h}_{i+1} = f_{W_i}(\boldsymbol{h}_i) = s(W_i\boldsymbol{h}_i), \quad i = 1, 2, \cdots, M-1 \quad (2)$$

where M is the number of layers in the network; $\boldsymbol{h}_i$ is the state (or the output signal) of i-th layer, which is a column vector. For example, $\boldsymbol{h}_1$ is the input layer and has the same dimension as input training data $\underline{\boldsymbol{x}}$. $\boldsymbol{h}_M$ is the output layer and has the same dimension as output training data $\underline{\boldsymbol{y}}$. $f_{W_i}(\cdot)$ is the feedforward mapping from the i-th to $(i+1)$-th layer, which is usually defined by a non-linear activation function $s(\cdot)$ with corresponding weights $W_i (i = 1, 2, \cdots, M-1)$. In this article, sigmoid function is used as the activation function. The weights W_i on i-th layer is a matrix whose dimension is d_{i+1}-by-d_i, where d_i is the number of neurons on i-th layer. For the simplicity of notation, bias term is not introduced in our model because it can easily be incorporated into the weights W_i. Denote $\Theta_W^{i,j}$ as the collection of all the weights between i-th and j-th layer.

$$\Theta_W^{i,j} = W_k, k = i, i+1, \cdots, j-1 \quad (i < j) \quad (3)$$

Therefore, the state of j-th layer can be written as

$$\boldsymbol{h}_j = \boldsymbol{h}_j(\boldsymbol{h}_i, \Theta_W^{i,j}) \quad (4)$$

III. Backpropagation

Backpropagation is a typical means of first-order algorithm for the training of feedforward neural network.

The objective network should give an output signal as close to the target output as possible when corresponding input sample is shown. The criteria of *close*, the loss function, is defined by the distance between target output and real output. It is written as $L(\boldsymbol{h}_M(\underline{\boldsymbol{x}}, \Theta_W^{1,M}), \underline{\boldsymbol{y}})$ which is the loss function measuring the accuracy of the network for a given sample $(\underline{\boldsymbol{x}}, \underline{\boldsymbol{y}})$. Therefore, the training objective is to minimize the global loss $\mathbb{E}\{L(\boldsymbol{h}_M(\underline{\boldsymbol{x}}, \Theta_W^{1,M}), \underline{\boldsymbol{y}})\}$ under data distribution $\{(\underline{\boldsymbol{x}}, \underline{\boldsymbol{y}})\}$. In this article, MSE is used as the loss function.

By taking the derivative iteratively, the gradient of loss function in terms of weight matrix on i-layer can be written as

$$\frac{\partial L}{\partial W_i} = 2(\boldsymbol{h}_M - y)\boldsymbol{h}_M(1-\boldsymbol{h}_M)\left(\prod_{k=i+1}^{M-1} W_k\boldsymbol{h}_k(1-\boldsymbol{h_k})\right)\boldsymbol{h}_i^T \quad (5)$$

Especially, for the last layer,

$$\frac{\partial L}{\partial W_{M-1}} = 2(\boldsymbol{h}_M - y)\boldsymbol{h}_M(1-\boldsymbol{h}_M)\boldsymbol{h}_{M-1}^T \quad (6)$$

Here, batch mode training can be adopted where the whole training data is divided into multiple batches and the weights of the network is updated each time one batch of data pair $(\underline{\boldsymbol{x}}, \underline{\boldsymbol{y}})$ is utilized. The weights is updated by following formula.

$$W_i^{(t)} = W_i^{(t-1)} - \alpha\frac{\partial L}{\partial W_i^{(t-1)}} \quad (7)$$

The coefficient α controls the speed of gradient descent and it should be a positive real number.

IV. RACOS

Randomized coordinate shrinking algorithm, a recently proposed zeroth-order method, is a means of classification-based optimization. An ordinary classification-based optimization algorithm is presented in Algorithm 1.

It starts from a uniformly generated set from the solution space, and then follow T iterations. In each iteration, a binary classification data set $B_t = \{(x, y)\}$ is formed. x is from the population generated by last generation and y_i is defined that

Algorithm 1 classification-based optimization

Collect $S_0 = \{x_1, \cdots, x_m\}$ by i.i.d. sampling from $\mathcal{U}_X$;
let $\tilde{x} = arg\min_{x \in S_0} f(x)$;
for $t = 1$ to T **do**
    Construct $B_t = \{(x_1, y_1), \cdots, (x_m, y_m)\}$, where $x_i \in S_{t-1}$ and $y_i = sign[\beta - rank_{S_{t-1}}[f(x_i)]]$
    let $S_t = \emptyset$
    for $i = 1$ to m **do**
        $h_i = \mathcal{C}(B_t)$, where $h_t \in \mathcal{H}$
        $x_i = Sampling(h_t, \lambda)$, and let $S_t = S_t \cup \{x_i\}$
    end for
    $\tilde{x} = arg\min_{x \in S_t \cup \{\tilde{x}\}} f(x)$
end for
return $\tilde{x}$ and $f(\tilde{x})$

(a) equals to 1 if x_i is among the best $\lfloor \beta \rfloor$ solutions; otherwise (b) equals to -1. The function $rank_{S_{t-1}}[f(x_i)]]$ returns n if $f(x_i)$ is the n-th minimum among all the values of $f(x)$, $\forall x \in S_{t-1}$. Afterwards, m individuals are generated one by one by $Sampling$ and added to the population for the next iteration.

h_t is the hypothesis given by the binary classification algorithm $\mathcal{C}$ based on the binary data set B_t, where hypothesis is a function mapping solution space X to $\{+1, -1\}$. Donate $D_h = \{x \in X | h(x) = +1\}$. The hypothesis h_t given by binary classification algorithm has to satisfy that $h_t(x) = +1, \forall (x, +1) \in B_t$ and $h_t(x) = -1, \forall (x, -1) \in B_t$.

Afterwords, x_i is generated by the $Sampling$ procedure, $x_i = Sampling(h_t, \lambda)$, where $\lambda \in [0, 1]$ is the balancing parameter, and it samples with probability λ uniformly from D_{h_t} and with remaining probability uniformly from the whole solution space X.

Among numeral approaches that can serve as the binary classification algorithm, *randomized coordinate shrinking* classification is adopted. The positive class region of the hypothesis h in *RACOS* algorithm remains a hypercubic whose edges are parallel to the axes, which shrinks quickly to cover all the positive examples but no negative ones.

RACOS classification algorithm is presented in Algorithm 2. In each run of this classification algorithm, B_t, a set of solutions with their objective values is given as the input. Initially, a positive example is randomly selected from the positive examples of B_t (Line 3) and the positive region of the hypothesis D_{h_t} is set to the solution space X (Line 4). Iteratively, the negative examples which is in D_{h_t} are selected one by one and the region is shrunk to exclude the selected negative example (Line 5-17). Besides, the number of uncertain bits of dimension is controlled and reduced to the specified number N. If there are more uncertain bits than expected, some uncertain bits are randomly selected and their values are constrained to be the values of corresponding bits of the previous selected positive sample (Line 18-21).

RACOS optimization algorithm is obtained by equipping this classification algorithm (Algorithm 2) into the ordinary classification-based optimization framework (Algorithm 1).

Algorithm 2 The *randomized coordinate shrinking* classification algorithm for $X = [0, 1]^n$

1: B_t^+ =the positive solutions in B_t
2: B_t^- =the negative solutions in B_t
3: Randomly select $x_+ = (x_+^{(1)}, \cdots, x_+^{(n)})$ from B_t^+
4: let $D_{h_t} = X$, $I = \{1, \cdots, n\}$
5: **while** $\exists x \in B_t^- : h_t(x) == +1$ **do**
6: k = randomly selected index from the index set I
7: $I = I - \{k\}$
8: x^- = randomly selected solution from B_t^-
9: $B_t = B_t - \{x^-\}$
10: **if** $x_+^{(k)} \geq x_-^{(k)}$ **then**
11: r = uniformly sampled value in $(x_-^{(k)}, x_+^{(k)})$
12: $D_{h_t} = D_{h_t} - \{x \in X | x^{(k)} < r\}$
13: **else**
14: r = uniformly sampled value in $(x_+^{(k)}, x_-^{(k)})$
15: $D_{h_t} = D_{h_t} - \{x \in X | x^{(k)} > r\}$
16: **end if**
17: **end while**
18: **while** $\#I > N$ **do**
19: k = randomly selected index from the index set I
20: $D_{h_t} = D_{h_t} - \{x \in X | x^{(k)} \neq x_+^{(k)}\}$, $I = I - \{k\}$
21: **end while**
22: **return** h_t

When the algorithm is applied to the training of multilayer feedforward network, a vector formed by all the elements of the weights in the whole network is optimized. That is $\boldsymbol{x} = \boldsymbol{\theta}_W^{1,M}$, where $\Theta_W^{1,M}$ is rearranged and formed as a vector $\boldsymbol{\theta}_W^{1,M}$. Therefore, the dimension of this optimization problem is $\sum_{i=1}^{M-1} d_i d_{i+1}$, where d_i is the number of neurons on i-th layer. The objective function $f(\boldsymbol{x}) = f(\boldsymbol{\theta}_W^{1,M})$ can be defined as the global loss with regard to given dataset $\{(\underline{\boldsymbol{x}}, \underline{\boldsymbol{y}})\}$.

It has to be noticed that the solution space in *RACOS* has an absolute boundary and can be designated as $[0, 1]^n$ after proper scaling, whereas in the case of the weights of a feedforward network, there is no specified boundary. However, since it is continuous optimization, the results will not be deteriorated if an arbitrary boundary which is big enough is imposed. Empirically, $[-4\sqrt{\frac{6}{d_i + d_{i+1}}}, 4\sqrt{\frac{6}{d_i + d_{i+1}}}]$ is used as the boundary of weights on i-th layer.

V. Hybrid Strategies

RACOS is applicable for optimization of complex functions such as the training of feedforward network. However, it ignores the structure of the feedforward network, loses gradient information and thus has low efficiency. Backpropagation converge faster than *RACOS*, but it can be trapped in local minima and misled by the gradient.

In order to inherit the merits of the both and compensate for the shortcomings, *BP* is designed to incorporate into *RACOS* and serve as inner evaluator, inner optimizer and/or outer optimizer.

BP can be incorporated as inner evaluator. Instead of evaluating the loss for potential solution generated by *RACOS* directly, several iterations of *BP* is firstly applied to the generated potential solution and then the updated solution is used to evaluate corresponding loss. Using *BP* as inner evaluator can exploit the gradient information around the solution generated by *RACOS*. This inner evaluator gives a good evaluation if the evaluated solution and its neighborhood are both satisfying. This kind of incorporation is designated as *RACOS(BP)*.

BP can also be incorporated as inner optimizer. In *RACOS*, the potential solution for the next generation $\boldsymbol{x}$ is sampled from the latest hypothesis in every epoch (Algorithm 1 Line 8). When *BP* is used as inner optimizer, the solution population is updated by BP once it is sampled from latest hypothesis. In another word, the solution is updated by *BP* and *RACOS* alternately epoch by epoch. By alternately applying zeroth- and first-order method, it will not only converge faster for the use of gradient information but also avoid local minima for the use of zeroth-order method. This is designated as *RACOS*BP*.

Again, *BP* can be incorporated as post-optimizer. Likewise, epochs of *BP* is applied to the solution found by *RACOS*. *RACOS*, as a type of zeroth-order method, which is not focusing on the local situation and not misled by the gradient information, can find a solution close to the global minima faster. However, the found solution usually does not lie exactly on the minima. Therefore, epochs of *BP* is needed to find the minima. This strategy is denoted as *RACOS+BP*.

The above three ways of incorporating *BP* into *RACOS* can be adopted in combination. Thus forming two more strategies - *RACOS(BP)+BP* and *RACOS*BP+BP*. The former one is applying epochs of *BP* to the solution found by *RACOS(BP)*, whereas the latter one is applying *BP* to the solution found by *RACOS*BP*. The reason why *BP* inner evaluator and *BP* inner optimizer are not combined is that using *BP* as inner optimizer is actually spontaneously indicating the incorporation of *BP* as inner evaluator.

To sum up, five hybrid strategies are proposed - that is *RACOS(BP)*, *RACOS*BP*, *RACOS+BP*, *RACOS(BP)+BP* and *RACOS*BP+BP*. Meanwhile, two single strategies, *BP* and *RACOS*, are also adopted as comparison with the above five in later experiment.

In order to measure and compare different strategies where zeroth- and first-order method are both applied, the cost of the hybrid strategies must be cleared. The computational cost is defined as follows. Either one query for the global error of the feedforward neural network in *RACOS* or one iteration of *BP* is defined as 1 unit of cost, since each of them involves iterating through all the training data once. The global loss on testing dataset is used as the merit function in the end.

Since as for *RACOS+BP*, *RACOS(BP)+BP* and *RACOS*BP+BP* the number of epochs of *BP* appended to the former hybrid step is not specified, their costs are hard to measure and compare. The ratio of concoction η is introduced and defined as

$$\eta(\text{algo + BP}) = \frac{\text{the cost of algo}}{\text{total cost}} = \frac{\text{the cost of algo}}{\text{the cost of algo+the cost of BP}} \tag{8}$$

For example, if the total cost is 30000 units, $\eta(RACOS + BP) = 0.5$ indicates that half of the total cost (that is 15000) is used by *RACOS* and the other half is used by *BP*.

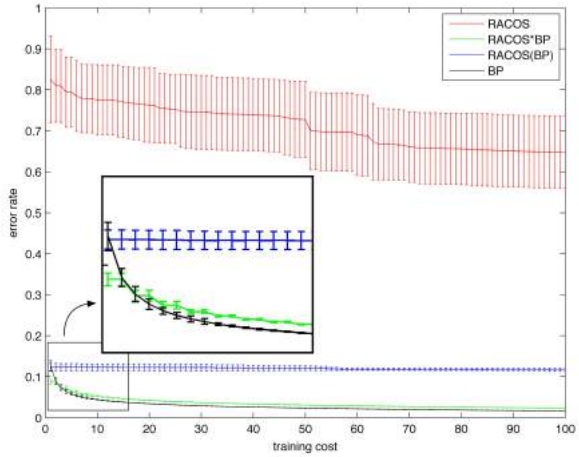

Fig. 3. Decrease of error over training cost for MNIST dataset on a network structured as $[784, 100, 100, 10]$.

VI. Experiments

The comparison of the strategies are carried out on MNIST dataset, which is a database of handwritten digits. It gives training data $(\underline{\boldsymbol{x}}, \underline{\boldsymbol{y}})$, where $\underline{\boldsymbol{x}}$ is the input training data whose dimension is $28 \times 28 = 784$ representing for an image of handwritten digits, and $\underline{\boldsymbol{y}}$ is the output training data whose dimension is 10, representing for the possibility of the ten digits.

This classification problem is examined by a 4-layer feed-forward network, where there is one layer as input layer whose dimension $d_1 = 784$, one as output layer whose dimension $d_4 = 10$, and two as hidden layers whose dimension $d_2 = d_3 = 100$. Hence, the network structure is written like $[784, 100, 100, 10]$.

All the results are tested parallelly 60 times with mean and standard deviation calculated. MNIST dataset is divided where 85% of the data is for training and 15% of the data is for testing. The error rate shown below is calculated from testing data.

Figure 3 is the graph of the decrease of error over training cost for MNIST dataset on above-mentioned feedforward network. Four strategies are examined - *RACOS*, *RACOS*BP*, *RACOS(BP)* and *BP*. Why these four are firstly presented is because they do not involve the ratio η. The x axis is the cost of the training and the y axis is the error rate on the testing data from MNIST dataset. For clarity of demonstration, here show only first 100 epochs. The comparison of *RACOS* with

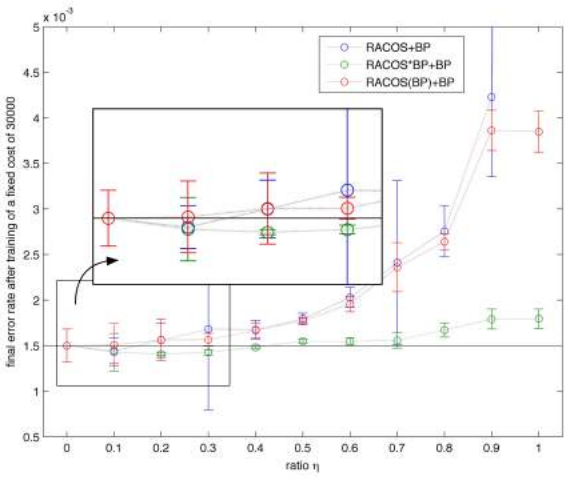

Fig. 4. The final error rate after a fixed training cost of 30000 when different ratios η are adopted. The black line horizontal to x axis is the reference line whose value equals to that of pure *BP*. The last point of *RACOS+BP* ($\eta = 1$) is out of plot range and not shown.

*RACOS*BP* and *RACOS(BP)* clearly suggests that the adoption of *BP* can promote the performance of *RACOS*. The adoption of *BP* as inner optimizer (*RACOS*BP*) is the most effective among these four. It is competitive with BP and shows better performance in the beginning epochs of the training.

Figure 4 is the final error rate after a training of fixed cost of 30000 when different ratio η are adopted. Notably, $\eta(RACOS + BP) = 0$ indicates that only pure BP is adopted whose final error rate is shown by the left end point. Therefore, left end point in the graph can serve as a reference point to examine the performance of hybrid strategies in comparison with pure BP. The black line horizontal to x axis is the reference line whose final error rate equals to pure *BP*, indicating that the points below this line have better performance than pure *BP*.

The result (especially the points inside the black box) shows that some hybrid strategies do better than pure *BP* when the ratio η is properly configured. *RACOS+BP* with $\eta = 0.1$ shows better performance over pure *BP*. Since the ratios η is quite small, *RACOS* can be regarded as preprocessing of *BP*. The meaning of this preprocessing is to find a more potential starting point for *BP* and avoid being trapped into local minima which often happens if the starting point is randomly generated. *RACOS*BP+BP* with η equivalent to 0.1 or 0.2 shows even better performance, which indicate that *RACOS*BP* is a more effective algorithm for preprocessing.

Table I concludes the best performance of each strategy after a training of fixed 30000 units of cost, where *RACOS*BP + BP* with $\eta = 0.2$ is the best (highlighted in bold).

Although only *BP* is being used as the representative of first-order method in our experiment, we believe that the performance of other first-order methods will also be improved if such zeroth-order preprocessing is being applied.

TABLE I
BEST PERFORMANCE OF EACH STRATEGY AFTER A TRAINING OF FIXED COST OF 30000

Algorithm	Final error rate
RACOS	58.523×10^{-3}
*RACOS*BP*	2.124×10^{-3}
RACOS(BP)	10.523×10^{-3}
RACOS+BP	1.442×10^{-3}
*RACOS*BP+BP*	$\mathbf{1.408 \times 10^{-3}}$
RACOS(BP)+BP	1.500×10^{-3}
BP	1.501×10^{-3}

VII. CONCLUSION

The training of feedforward neural network is decomposed as an optimization problem in nature and a zeroth-order method *RACOS* is successfully adopted to solve this problem, though this problem is canonically solved by first-order methods. Furthermore, incorporating backpropagation algorithm, several hybrid strategies are proposed, examined and compared. The hybrid strategies inherit the merits of zeroth- and first-order method and compensate for the shortcomings of both, thus presenting features like fast convergence and avoidance of being trapped into local minima. All of the hybrid strategies promote the performance of pure *RACOS*, and one of them shows better performance than conventional *BP* algorithm. This is a successful attempt of combining zeroth- and first-order method to the training of feedforward neural network and it is open to further exploring.

REFERENCES

[1] T. Bäck, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford, UK: Oxford University Press, 1996.

[2] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford, U.K.: Oxford University Press, 1995.

[3] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[4] Y. Yu, H. Qian, and Y.-Q. Hu, "Derivative-free optimization via classification," in *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, Phoenix, AZ, 2016.