Computers & Industrial Engineering 158 (2021) 107400

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

Random mask-based estimation of the distribution algorithm for stacked auto-encoder one-step pre-training

Qingyang Xu [a,*], Anbang Liu [a], Xianfeng Yuan [a], Yong Song [a], Chengjin Zhang [a], Yibin Li [b]

[a] School of Mechanical, Electrica & Information Engineering, Shandong University, Shandong 264209, China
[b] School of Control Science and Engineering, Shandong University, Shandong 250100, China

ARTICLE INFO

Keywords:
Deep neural network
Stacked auto-encoder
Estimation of distribution algorithm
Random mask
One-step pre-training

ABSTRACT

The deep learning techniques have received great achievements in computer vision, natural language processing, etc. The success of deep neural networks depends on the sufficient training of parameters. The traditional way of neural network training is a gradient-based algorithm, which suffers the disadvantage of gradient disappearing, especially for the deeper neural network. Recently, a heuristic algorithm has been proposed for deeper neural network optimization. In this paper, a random mask and elitism univariate continuous estimation of distribution algorithm based on the Gaussian model is proposed to pre-train staked auto-encoder, and then a Stochastic Gradient Descent (SGD) based fine-tuning process is carried out for local searching. In the improved estimation of the distribution algorithm, two individual update strategies are defined; one group of individuals is generated according to the constructed probabilistic model, and another is updated according to the statistics of advanced individuals that aim to reduce the probability of combination explosion and time consumption according to the mask information. In the simulations, different architectures, different mask ratios and different promising individual ratios are adopted to testify the effectiveness of the improved algorithm. According to simulation results, the estimation of thr distribution algorithm has a steady optimization ability for the shallow and stacked auto-encoder by one-step pre-training combining SGD based fine-tuning for the MNIST dataset. The proposed model will achieve a state-of-the-art performance on Fashion-MNIST.

1. Introduction

Artificial intelligence (AI) has developed fast recently, driven by deep learning technology. Deep learning is an extension of the classic Artificial Neural Network that takes advantage of the current computational capability of the computer as well as the big data technology. Deep Neural Network (DNN), which is a nonconvex model, has become a powerful and extremely popular technique widely used for various nonlinear problems, such as computer vision and image recognition (LeCun et al., 2015), time series forecasting (Kuremoto et al., 2014; Wang et al., 2017) and video recognition (Martín et al., 2018). Back-propagation algorithm is widely used in neural network training, which relies on gradient information. It is effective for the shallow neural network. However, the gradient-based algorithms have several limitations, i.e., the algorithms suffer from gradient disappearing, easily affected by the initial hyper-parameters and easily trapped in local optima (Ye, 2017). Some improved algorithms are proposed, such as Stochastic Gradient Descent (SGD), AdaGrad, RMSProp, AdaDelta, etc. Hinton (Hinton and Salakhutdinov, 2006) proposed that with the pre-training technique, a deep neural network can be trained better based on the current and follow-up research. However, it is also based on the gradient algorithm, and the pre-training strategy only provides a better initial weight of deep architectures.

The successful applications of deep neural networks drive researchers to develop new methods and tools for parameters and architecture optimization. The topologies and training hyper-parameters are traditionally solved by manual initialization and the weights are trained by the gradient methodology. The training of deep neural networks depends crucially on the specifications of hyper-parameters, such as architecture, learning rate, regularization parameter, weight and many others (Hinz et al., 2018). The parameter optimization of a deep neural network can be described as an Eq. (1).

$$\mathrm{A} = \arg\min_{m_{\alpha,\Lambda,w}} L\left(m_{\alpha,\Lambda,w}\right) + R(w) \tag{1}$$

where A is an optimal deep neural network, $m_{\alpha,\Lambda,w}$ is the set of general

* Corresponding author.
E-mail address: qingyangxu@sdu.edu.cn (Q. Xu).

https://doi.org/10.1016/j.cie.2021.107400

Available online 17 May 2021

Table 1
Different heuristic algorithm-based deep neural network optimization.

	Optimization algorithm	Hyper-parameter	Architectures searching	Weights searching
Stanley (Snoek et al., 2012)	Genetic algorithm		√	√
Assun et al. (Assunção et al., 2019)	Genetic algorithm		√	
Snoek et al. (Bello et al., 2017)	Bayesian	√		
Bello et al. (Zoph et al., 2018)	Reinforcement learning		√	
Zoph et al. (Zhong et al., 2018)	Random search		√	
Zhong et al. (Real et al., 2017)	Reinforcement learning		√	
Real et al. (Xie et al., 2018)	Genetic algorithm		√	
Xie et al. (Cai et al., 2018)	Reinforcement learning		√	√
Cai et al. (Zhang et al., 2018)	Reinforcement learning		√	
Zhang et al. (Sun et al., 2020)	Random sampling		√	
Junior et al. (Junior and Yen, 2019)	Particle swarm optimization		√	
Sun et al. (Sun et al., 2019; Xu et al., 2019)	Genetic algorithm		√	√
Xu et al. (Lopez-Rincon et al., 2018)	Reinforcement learning	√		
Bergstra (Bergstra and Bengio, 2012)	Random search	√		
Lopez-Rincon et al. (Baldominos et al., 2018)	Genetic algorithm	√		
Baldominos et al. (Miuhlenbein and Paaß, 1996)	Genetic algorithm	√		
Martin et al. (Martín et al., 2018)	Genetic algorithm	√	√	

deep neural networks, α is hyper-parameter of training, Λ is the architecture of a deep neural network, w is the weight and $R(w)$ is regularization term. $L()$ is a loss function which is negative cross-entropy for classification problem. Therefore, the optimization of deep neural networks completely or partially includes α, Λ, w optimization, which is a large-scale optimization problem.

Some recent results indicate that some automated approaches can find better hyper-parameters and achieve better results than humans (Assunção et al., 2019; Junior and Yen, 2019; Bergstra and Bengio, 2012; Sun et al., 2019; Zhang et al., 2019; , 2020). However, it is a challenging work for large-scale weight optimization, resulting in high-dimensional searching space. Estimation of distribution algorithm is a kind of evolutionary algorithm. Although there are some researches about higher dimensional problems optimization based on estimation of distribution algorithm, the scale of the problem is limited and doesn't reach the scale of deep neural network's parameters. However, the global optimization capability of estimation of distribution algorithm is in accordance with the concept of Hinton's layer-wise pre-training. Therefore, we proposed an estimation of the distribution algorithm based on the deep neural network weights training algorithm. In order to improve the large-scale optimization capability of estimation of distribution algorithm, a random strategy is introduced into the estimation of the distribution algorithm, and the individuals of estimation of distribution algorithm are updated by two strategies which may speed up the training and reduce the combinational explosion probability. The large-scale optimization capability of estimation of distribution algorithm is enhanced. A comparison between the two steps (layer by layer) and one-step (pre-training all of the hidden layers) pre-training is done in the experiments, and the one-step pre-training is proved as effective and convenient for stacked auto-encoder pre-training based on the improved estimation of distribution algorithm. After the one-step pre-training, a fine-tuning process is carried out for local optimization. Compared with state-of-the-art results, the proposed algorithm is promising.

The rest of this paper is organized as follows: A literature review is presented in Section 2. The stacked auto-encoder is presented in Section 3. The proposed algorithm is described in Section 4. The experimental design is presented in Section 5. Finally, Section 6 presented the conclusion.

2. Literature review

Grid search and random search are two of the most widely-used approaches in parameter optimization (Ma et al., 2020). In grid search, a pre-determined range of hyper-parameters is defined to construct every possible combination of hyper-parameter. However, the grid grows exponentially with the number of parameters. On the other hand, random search generates a random value from a pre-defined distribution of the parameter. Bergstra and Bengio (Ma et al., 2020) show that random search performs almost or equally well as the grid search empirically in higher dimensional spaces and is much quicker. Heuristic methods are also introduced for parameter optimization (Guo et al., 2020; Oong and Isa, 2011). Evolutionary algorithms (EAs) have good global search capabilities that are likely to provide the most promising solution, especially for nonconvex problems (Al-Dabbagh et al., 2015; Al-Dabbagh et al., 2015; Yao and Liu, 1997; Hussain et al., 2019; Lv et al., 2019; Hu and Yang, 2019; Wu et al., 2020; Wu et al., 2016; Wu et al., 2016; Wu et al., 2018; Cheng et al., 2020; Stanley and Miikkulainen, 2002). Table 1 is the deep neural network optimization approaches based on EA recently. According to Table 1, the studies mostly aim to evolve the topologies of the deep neural network due to the challenges of determining its complexity. The topologies of artificial neural networks are encoded as a chromosome in the evolutionary algorithm, and then some operators are adopted for evolution. Therefore, the choice of variable is defined by some common or hand-crafted architectures, such as convolution, pooling, skip connections, etc. However, the optimization of weights is always based on a gradient algorithm due to the scale of the problem.

According to Table 1, these researches are almost based on a genetic algorithm, which makes use of various kinds of operators, such as selection, crossover, and mutation, to produce offspring. The population modeling-based evolutionary algorithms are rarely seen in Table 1, such as estimation of distribution algorithms (Dong et al., 2013), which makes use of promising individuals from the current population to construct a probabilistic model (instead of using crossover or mutation operators) for offspring generation. Genetic algorithms allow a direct operation of individuals and obtain the gene of the best solutions found so far, whereas the estimation of distribution algorithms have indirect operations of offspring that make use of the probabilistic model as guidance for population reproduction to find better solutions (Sun et al., 2020). This operation grasps the trend of population evolution. The estimation of distribution algorithms has been shown to experimentally outperform other existing algorithms on many benchmarks and has been applied to various fields (Wang et al., 2015; Pérez-Rodríguez and Hernández-Aguirre, 2019; Arin and Rabadi, 2017; Wang et al., 2012; Chen et al., 2010). The performance of estimation of distribution algorithm depends on how well is the probabilistic model learned (Mishra and Gallagher, 2014). However, there were experimental observations that estimation of distribution algorithms did not scale well to large-scale problems since probabilistic model construction is the foundation of estimation of distribution algorithms. As the truncation selection is always adopted in the estimation of distribution algorithms (Hansen and Ostermeier, 2001), they must suffer from the well-known curse of

dimensionality (Sun et al., 2020). According to the curse of dimensionality theory, the quantity of data will increase exponentially with the dimensionality of search space in order to maintain a proper spatial density. The estimation of distribution algorithms tries to learn the global statistical information from some promising samples. Therefore, the population size of estimation of distribution algorithms has to grow quickly with the problem size in order to maintain a good performance. This is a challenge for the large-scale problem and optimal probabilistic model learning, such as weights of deep neural networks. Therefore, the performance of estimation of distribution algorithm on higher dimensional problems (e.g., 500-D) is rarely studied (Sun et al., 2020). There have been only a few attempts on large-scale (≥500-D) problems. Gaussian model suffers less from the curse of dimensionality than the histogram model, because the Gaussian model usually has fewer degrees of freedom. CMA-ES (Wang et al., 2008) adopted a covariance matrix adaptation (CMA) for population generation. The diagonal covariance matrix relies on the cumulation of the evolutional population, which can reduce the population size. Wang and Li (Bielza et al., 2009) proposed a univariate model-based estimation of distribution algorithm LSEDA-gl for large-scale optimization benchmarks. Mixed Gaussian and Levy probability distribution, standard deviation control strategy and restart strategy were adopted for sampling. However, the general performance of broader types of high-dimensional problems is still unknown. Bielza et al. (Karshenas et al., 2013) made use of larger genes and smaller samples for the classification problem using logistic regression regularizers. In ref. (Bosman, 2009), regularization techniques were introduced into the estimation of the distribution algorithm, which displayed the ability to solve high dimensional problems with a comparable quality of solutions and a smaller population. Bosman (Kabán et al., 2016) proposed a Gaussian estimation of distribution algorithm AMaLGaM which was used for 1000-D problems, and the memory techniques were adopted. EDA Model Complexity Control EDA-MCC (Sun et al., 2020) is the first attempt at scaling up the multivariate model-based estimation of distribution algorithm for large-scale continuous optimization (up to 500-D problems) by weakly dependent variable identification and subspace modeling. RP-EDA (Omidvar et al., 2017), assemble random projection to the estimation of distribution algorithm to find approximate solutions, generate a random projection matrix and project the center point into k dimensions to estimate the k*k sample covariance for new sample generation. DECC-DG (Hauschild and Pelikan, 2011) adopted differential grouping decomposition for large-scale global optimization problems, which is a type of Cooperative Co-evolution approach, and DG2 found a reliable threshold value by estimating the magnitude of round off errors, reused the sample points detecting interactions, and saved up to half of the computational resources on fully separable functions. However, the general performance of these studies on large-scale problems is still unknown, such as for the 10,000-D, 100,000-D and 1,000,000-D.

We built upon the univariate model-based estimation of the distribution algorithm, and incorporated the random strategy for the population updating, which is used for deep neural network weights optimization. The population updating strategy aims to improve the optimization efficiency and reduce the probability of a combinational explosion. In summary, the contribution of this paper is as following: (1) the proposed model in this study is used for deep neural network weight optimization which is a huge large-scale optimization problem; (2) compared with the existing estimation of distribution algorithm, the proposed model makes use of random updating strategy for population generation; (3) based on the proposed model, the one-step pre-training strategy is effective for the deep neural network pretraining.

3. Stacked Auto-encoder

Hinton proposed the layer-wised weight initialization method to gather a rough region of the weight, and then a fine-tuning process is carried out to search the solution accurately. If the initial weights are close to a good solution, the gradient descent algorithm works well. Auto-encoder is a simple network for deep neural network pre-training. It is exemplified in Fig. 1.

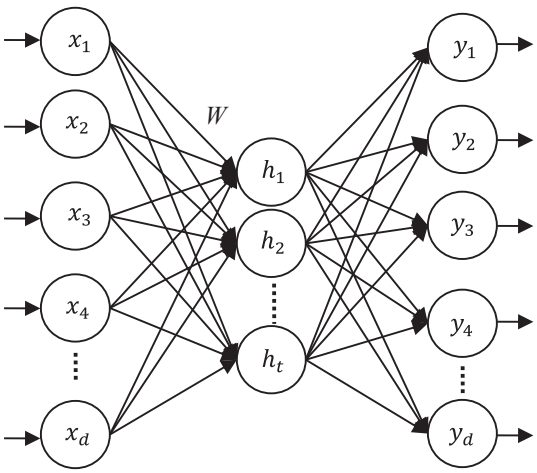


Fig. 1. Diagram of the Auto-encoder.

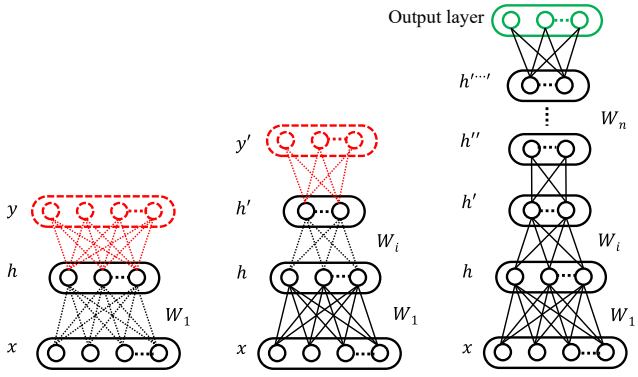


Fig. 2. Principle of stacked auto-encoder training.

The auto-encoder takes an input $x \in R^d$ and maps it to a latent representation space $h \in R^t$ by a single-layer neural network. The encoder transforms input x into latent representation h by equation (2):

$$h = \sigma(WX) \tag{2}$$

where W is the weight matrix between input and output neurons and σ is a sigmoid function.

For the decoding procedure, the hidden representation h is then mapped back to a reconstructed vector y by $g(h) = \sigma(W^{'}h)$. This affine mapping is called a decoder. In general, y is not to be interpreted as an exact reconstruction of x, but rather in probabilistic terms, as the parameter of a distribution $p(X|Y = y)$ that may generate x with high probability.

The deep neural network has an input layer, two or more hidden layers, and an output layer. The hidden layers can be pre-trained in an unsupervised way using an auto-encoder, starting from the first hidden layer, and then going layer by layer. Once a given auto-encoder has been trained to reconstruct the input successfully, the output layer is no longer needed and the corresponding hidden layer becomes the input layer of the next hidden one as shown in formula (3). The next hidden layer is also pre-trained as an independent auto-encoder, and the process is repeated until there are no more hidden layers. The $clf()$ is the classification function, such as Softmax. This is also called a stacked auto-encoder as shown in Fig. 2.

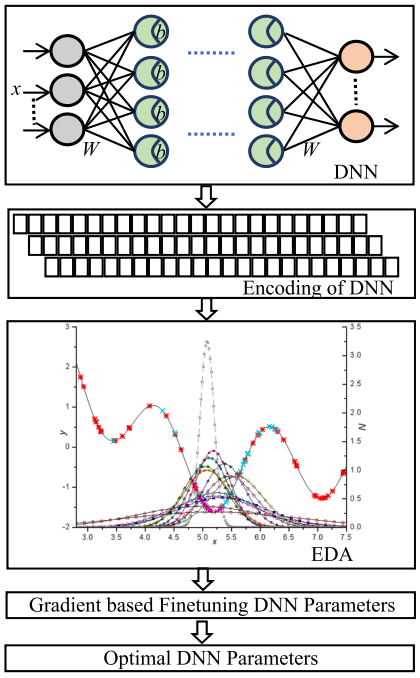


Fig. 3. Diagram of stacked auto-encoder optimization by EDA.

$$\begin{cases} h_1 = f(W_1 X) \\ h_2 = f(W_2 h_1) \\ \cdots \\ h_i = f(W_i h_{i-1}) \\ \cdots \\ h_n = f(W_n h_{n-1}) \\ o = clf(h_n) \end{cases} \tag{3}$$

The goal of unsupervised pre-training is to initialize the weights to a better region of parameter space than random initialization. Once a stacked auto-encoder has been built, the final hidden layer output representation can be used as input of a stand-alone supervised learning algorithm, for example, a Support Vector Machine classifier, a Softmax classifier, or a (multi-class) logistic regression. In this context, a subsequent supervised training process called fine-tuning can be achieved by conventional gradient descent. The classical squared error (MSE) loss function is used to minimize the error of practical output and target output as equation (4).

$$MSE = \frac{1}{n}\sum\left(x - y\right)^2 \tag{4}$$

In addition, some regularization terms can be adopted. Thus, the loss function turns out to be as shown in equation (5).

$$L = \frac{1}{n}\sum\left(x - y\right)^2 + \beta\sum Sparse\left(\rho\middle|\middle|\widehat{\rho}\right) + \lambda\sum W^2 \tag{5}$$

$$Sparse(\rho||\widehat{\rho}) = \rho log\,\frac{\rho}{\widehat{\rho}} + (1-\rho) log\,\frac{1-\rho}{1-\widehat{\rho}} \tag{6}$$

where $Sparse(\rho||\widehat{\rho})$ is a sparse penalty term, $\widehat{\rho}$ is average activation of hidden unit, ρ is a sparsity value, and β is the weight of sparsity penalty term. λ is the coefficient of weight regularization. The pseudocode of auto-encoder training is shown in algorithm 1.

Algorithm 1: Pseudocode of auto-encoder training

Initialization: Input data $\boldsymbol{x}$, the labels $\boldsymbol{t}$, and other hyper parameters.
Pre-training:
Create the first auto-encoder $\boldsymbol{AE}^1$ with the input layer and the first hidden layer
Train $\boldsymbol{AE}^1$ by input data $\boldsymbol{x}$ and the reconstructed data$\boldsymbol{y}$
Finish pre-training the first hidden layer

(*continued on next column*)

(*continued*)

Algorithm 1: Pseudocode of auto-encoder training

Pre-train the remaining hidden layer
Fine-tuning:
Add the output layer
Train stack $\boldsymbol{AE}$ by backpropagation according to$\boldsymbol{E} = \frac{1}{\boldsymbol{n}}\sum(\boldsymbol{x} - \boldsymbol{t})^2$

4. Improved EDA based pre-training for stacked auto-encoder

For estimation of distribution algorithm based stacked auto-encoder training, the weight of the neural network is encoded as an individual of estimation of distribution algorithm. The statistical information of each weight can be calculated to construct a probabilistic model according to the advanced individual. Then, the new individuals are generated by probabilistic model sampling. Finally, the pre-trained parameters can be obtained for the stacked auto-encoder. The diagram of optimization is shown in Fig. 3.

4.1. Estimation of distribution algorithm

Estimation of distribution algorithm is a kind of evolutionary algorithm based on population modeling. The central idea of estimation of distribution algorithm is to maintain an explicit probabilistic model to represent the distribution and subsequent generation of better candidate solutions, at the next step by probabilistic sampling (Li et al., 2012). A typical estimation of distribution algorithm pseudo code is shown in algorithm 2.

Algorithm 2: Pseudocode of traditional estimation of distribution algorithm

Initialization: Set iteration = 0, generate initial population
Evaluation: Evaluate objective function $\boldsymbol{E}(\boldsymbol{x})$ for individuals
Selection: Select promising individuals
Modeling: Build probabilistic model $\boldsymbol{P}(\boldsymbol{x})$ based on promising individuals
New population generation: Generate new population by sampling$\boldsymbol{P}(\boldsymbol{x})$
Iteration = Iteration + 1
Go to step 2 until a stopping criterion

N represents the population size, andBN is number of promising individuals. The initial population of estimation of distribution algorithm is always initialized with a random variable $z_i \in [a_i, b_i]$ as shown in equation (7).

$$x_i^n = L_i + \frac{H_i - L_i}{b_i - a_i}(z_i - a_i) \tag{7}$$

where x_n^i is i-th variable of n-th individual, z_i is a random variable, a_i and b_i are the bounds of i-th random variable, L_i and H_i are the bounds of i-th optimized variable.

The most important and crucial step of estimation of distribution algorithm is to construct a probabilistic model of an optimized variable. However, for the large-scale optimization problem, the computational cost of the multivariable joint probabilistic model is too high for the large-scale optimized variables and numerous individuals for each variable (Ahmed et al., 2018; Lu et al., 2019). The univariate model supposes that the variables have independent and identically distribution, and obey normal distribution. The probabilistic model of individuals for the estimation of distribution algorithm$P(x_1, x_2, \cdots x_D)$ is as following:

$$P(x_1, x_2, \cdots x_D) = \prod_{i=1}^{D} N(x_i|\mu_i, \sigma_i) \tag{8}$$

$(x_1, x_2, \cdots x_D)$ are variables to be solved, μ_i and σ_i are the mean and standard deviation of $i-$th variable, $N(x_i|\mu_i, \sigma_i)$ is normal probabilistic distribution model as following:

$$N(x_i|\mu_i, \sigma_i) = \frac{1}{\sigma_i\sqrt{2\pi}} e^{-\frac{(x_i-\mu_i)^2}{2\sigma_i^2}} \tag{9}$$

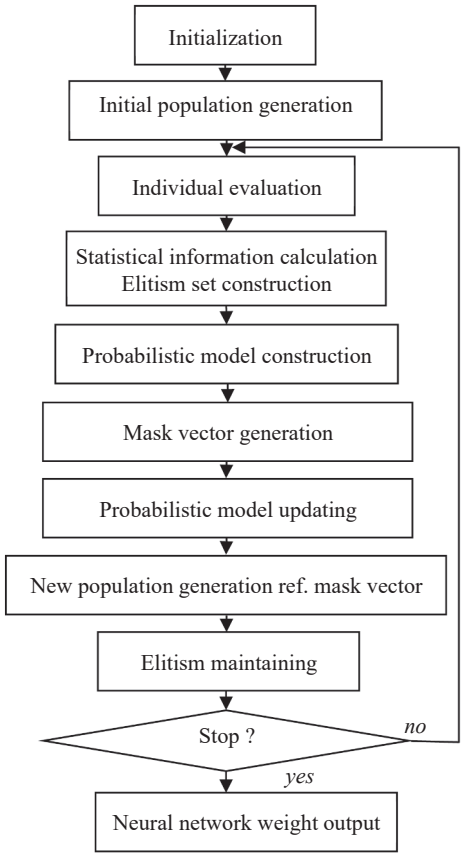


Fig. 4. Flowchart of improved EDA.

$P(x_1, x_2, \cdots x_D)$ can be described further as shown in equation (10).

$$P(x_1, x_2, \cdots x_D) = \prod_{i=1}^{D} \frac{1}{\sigma_i \sqrt{2\pi}} e^{-\frac{(x_i-\mu_i)^2}{2{\sigma_i}^2}} \tag{10}$$

(μ_i, σ_i) can be computed in a common way as follows:

$$\mu_i = \frac{1}{BN} \sum_{n=1}^{BN} x_i^n \tag{11}$$

$$\sigma_i^2 = \frac{1}{BN} \sum_{n=1}^{BN} \left(x_i^n - \mu_i(k)\right)\left(x_i^n - \mu_i\right)^T \tag{12}$$

where BN is the number of promising individuals, and n is the promising individual. At the final stage, the new population can be generated by sampling the model $P(x)$.

4.2. Improved estimation of distribution algorithm

4.2.1. Random mask strategy

For traditional univariate estimation of distribution algorithms, the new population is completely generated by sampling the probabilistic model of the corresponding variable. Therefore, the optimal solution is the combination of the optimum value of each variable, which is a continuous process (Srivastava et al., 2014). However, for a large-scale optimization problem, huge time consumption and combinatorial explosion maybe occur. Therefore, it is a challenge for traditional estimation of distribution algorithms. Kabán et al. (Omidvar et al., 2017) introduced random projections and a low dimensional fittest solution strategy to improve the large-scale optimization capability of estimation of the distribution algorithm. Moreover, inspired by the theory of dropout (Ahn and Ramakrishna, 2003), which deactivates neurons in the output layer of a neural network, a different individual updating strategy is adopted for variables. In this way, the problem of huge time consumption and combinatorial explosion can be reduced. The flowchart of the proposed estimation of the distribution algorithm is shown in Fig. 4.

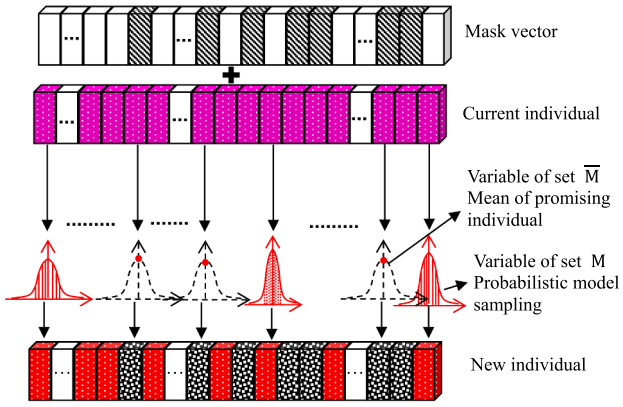


Fig. 5. Diagram of individual updating.

According to the flowchart of the improved estimation of distribution algorithm, individual updating is different from the traditional estimation of distribution algorithm. For this improvement, two variable updating strategies are proposed. Firstly, a mask vector is generated randomly within the number of variables, which is used to divide the variables set into M and $\overline{M}$ as reference for variable updating.M is defined according to the mask vector which is an integer set corresponding to the index of variables. Other variables outside of set M are defined as set $\overline{M}$.

$$P = \begin{cases} P_i(k) & x_i \in M \\ P_i(k-1) & x_i \in \overline{M} \end{cases} \tag{13}$$

where k is current iteration steps, $x_i(k)$ is i-th variable of individuals, $P_i(k)$ is the new probabilistic model of variable x_i which is updated by current promising individuals as equation (14), and $P_i(k-1)$ is the previous probabilistic model of variable x_i.

$$P_i(k) = \frac{1}{\sigma_i(k)\sqrt{2\pi}} e^{-\frac{\left(x_i(k)-\mu_{i(k)}\right)^2}{2{\sigma_i}^2(k)}} \tag{14}$$

The new population is generated by sampling the new probabilistic model partially, and others are replaced by the statistical value of promising individuals' variable as shown in formula (15).

$$x_i^n(k) = \begin{cases} Sam\left(P_i^n(k)\right) & x_i \in M \\ \mu_i & x_i \in \overline{M} \end{cases} \tag{15}$$

$x_i^n(k)$ is i-th variable of n-th individual, μ_i is mean value of promising individual of i-th variable, $Sam()$ is a sampling function. The diagram of individual updating is shown as Fig. 5, and the pseudocode of improved estimation of distribution algorithm is shown in algorithm 3.

Algorithm 3: Improved estimation of distribution algorithm

Initialization: Set iteration = 0, generate initial population
Evaluation: Evaluate objective function $\boldsymbol{E}(\boldsymbol{x})$ for individuals
Selection: Select promising individuals and construct an elite set
1. Calculation: Calculate the statistical information $(\boldsymbol{u_i}, \boldsymbol{\sigma_i})$ of promising individuals

$$\boldsymbol{\mu_i} = \frac{1}{\boldsymbol{BN}} \sum_{n=1}^{\boldsymbol{BN}} \boldsymbol{x_i^n \sigma_i^2} = \frac{1}{\boldsymbol{BN}} \sum_{n=1}^{\boldsymbol{BN}} \left(\boldsymbol{x_i^n} - \boldsymbol{\mu_i(k)}\right)\left(\boldsymbol{x_i^n} - \boldsymbol{\mu_i}\right)^T$$

2. Mask vector: Generate random vector randomly, and define $\boldsymbol{M}$ and $\overline{\boldsymbol{M}}$ set
3. Probabilistic model updating: Update probabilistic model $\boldsymbol{P}(\boldsymbol{x})$ by different strategies according to $\boldsymbol{M}$ and $\overline{\boldsymbol{M}}$ set

$$\boldsymbol{P} = \begin{cases} \boldsymbol{P_i(k)} & \boldsymbol{x_i} \in \mathbf{M} \\ \boldsymbol{P_i(k-1)} & \boldsymbol{x_i} \in \overline{\mathbf{M}} \end{cases}$$

4. New population generation: Generate new population by different strategies

(continued on next page)

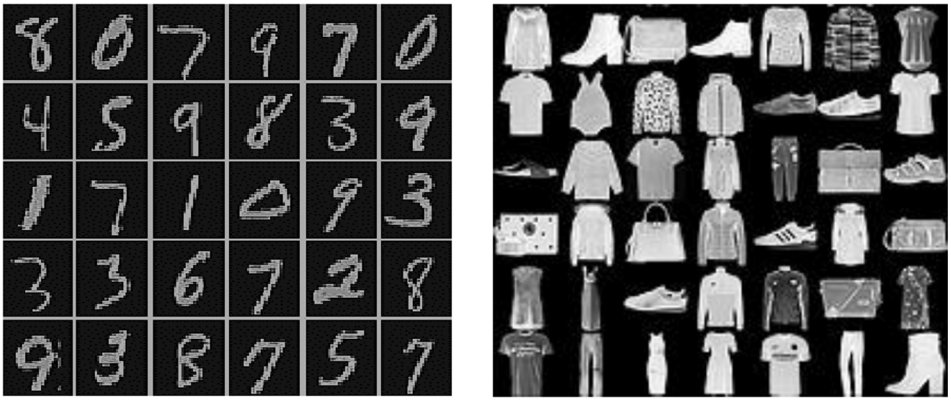


Fig. 6. Sample of MNIST and Fashion-MNIST.

(*continued*)

Algorithm 3: Improved estimation of distribution algorithm

$$x_i^n(k) = \begin{cases} Sam(P_i^n(k)) & x_i \in \mathbf{M} \\ \mu_i & x_i \in \overline{\mathbf{M}} \end{cases}$$

5. Elite maintaining
6. Iteration = Iteration + 1

Go to step 2 until a stopping criterion (such as max iteration steps)

4.2.2. Random mask strategy analysis

The weight optimization of stacked auto-encoder can be described as the minimization of energy function L.

$$A_w = \arg\min_w L(\Lambda_w) \tag{16}$$

where w is the weight of stacked auto-encoder, Λ is the neural network, and L is the energy function.

Supposing X is the input of a certain layer, and W is the weight of the hidden layer, Net_i is the sum of hidden layer input and f is the activation function. Therefore, the output of L-th layer and i-th neuron can be defined as follows:

$$a_i^L = f(Net_i) \tag{17}$$

$$Net_i = g_x(W) = WX \tag{18}$$

where,

$$W = \begin{pmatrix} w_{11} & \cdots & w_{1m} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & w_{nm} \end{pmatrix} \tag{19}$$

For the weight optimization problem, variable W is the optimization parameters and X is the given data.

According to the assumption of estimation of distribution algorithm, w_{ij} obeys normal distribution. Therefore, Net_i can be described as Eq. (20).

$$Net_i = \begin{bmatrix} Net_1 \\ \vdots \\ Net_n \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^m w_{1i}{}^* x_i \\ \vdots \\ \sum_{i=1}^m w_{ni}{}^* x_i \end{bmatrix} \begin{bmatrix} N\left(\sum_{i=1}^m \mu_{1i}{}^* x_i, \sum_{i=1}^m \sigma_{1i}{}^* x_i\right) \\ \vdots \\ N\left(\sum_{i=1}^m \mu_{ni}{}^* x_i, \sum_{i=1}^m \sigma_{ni}{}^* x_i\right) \end{bmatrix} \tag{20}$$

μ_{ij} and σ_{ij} are mean and standard deviation of weight w_{ij}.

Two updating strategies are adopted for weight w_{ij} in improved EDA. For example, 5% of weight w_{ij} is updated by probabilistic model sampling, and random mask strategy for others. Therefore, Net_i can be described as Eq. (21).

$$Net_i^{'} = \begin{bmatrix} Net_1^{'} \\ \vdots \\ Net_n^{'} \end{bmatrix} \begin{bmatrix} N\left(\sum_{jin\overline{M}} w_{1j}{}^* x_j + \sum_{iinM} \mu_{1i}{}^* x_i, \sum_{iin\overline{M}} \sigma_{1i}{}^* x_i\right) \\ \vdots \\ N\left(\sum_{jin\overline{M}} w_{nj}{}^* x_j + \sum_{iinM} \mu_{ni}{}^* x_i, \sum_{iin\overline{M}} \sigma_{ni}{}^* x_i\right) \end{bmatrix} \tag{21}$$

For set $\overline{M}$, w is a scalar, whereas w is a variable in set M that is generated by the sampling of a probabilistic model. When the elements of the set M and $\overline{M}$ are infinite, the mean value of Net_i and $Net_i^{'}$ are equivalent, and standard deviation of $Net_i^{'}$ is reduced to 5% of Net_i. Therefore, the risk of combination explosion is reduced. Additionally, in the sampling of a probabilistic model partially, the time consumption is reduced accordingly.

4.2.3. Elitism strategy

Elitism strategy is an effective strategy to ensure that the best individual(s) is selected as the next generation in evolutionary algorithms because the best individual(s) maybe include the genes of optimal solution (Purshouse and Fleming, 2002). Therefore, elitism strategy can improve the convergence performance of evolutionary algorithms in many cases (Gao and de Silva, 2018). It is achieved by simply copying the best individual(s) directly to the new generation (Rumelhart et al., 1986). However, the number of best individuals selected as the next generation must be handled properly and carefully; otherwise, it may lead to premature convergence or cannot improve the performance of the algorithm.

$$Pop(k) = Elite(d) \mapsto rand(PopTemp(n), d) \tag{22}$$

where $Elite(d)$ is an elitism maintaining function to select d elites, d is the elitism number, $Pop(k)$ is the k-th population, n is the population size, $rand(PopTemp(n), d)$ indicates the selection of d random individuals from the current population $PopTemp$. $\mapsto$ is a replacement operation to replace the d random individuals by d elites.

4.3. Fine-tuning of stacked auto-encoder

The pertaining process achieves rough region searching. However, the final solution should be fine-tuned in another way due to the weak local optimization capability of estimation of distribution algorithm (Wang et al., 2012). A fine-tuning process is carried out based on the gradient algorithm for a stacked auto-encoder. The gradient information is calculated and used for weight tuning as following equation (Xiao et al., (2017) arXiv/1708.07747.).

$$w = w - \eta \frac{\partial E}{\partial w} \tag{23}$$

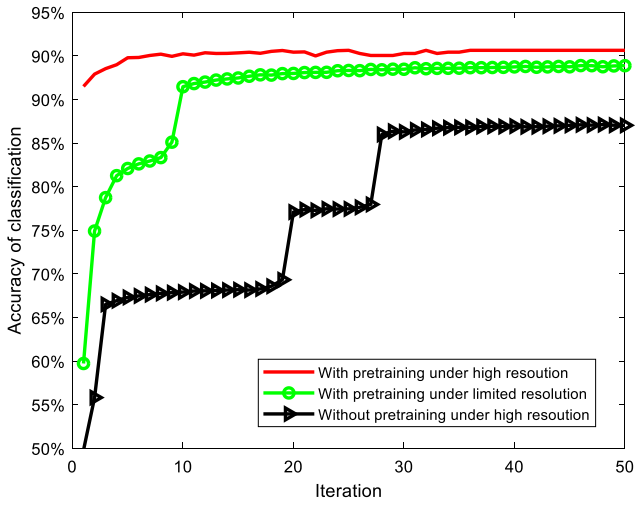


Fig. 7. Comparison of the effect of data resolution and pre-training on MNIST.

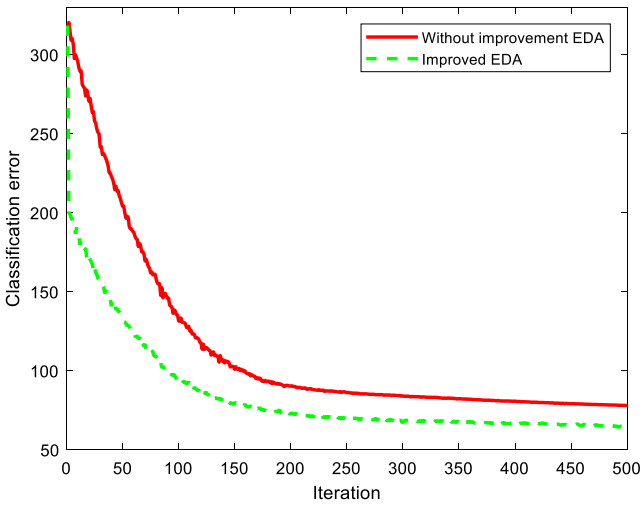


Fig. 8. Comparison of improved EDA with traditional EDA on MNIST.

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial net_j} x_j \tag{24}$$

where η is the learning rate, w is the weight, E is the loss value, x_j is the j-th neuron input from the previous layer, and net_j is the j-th neuron sum of input.

5. Experimental studies

To study the performance of the proposed algorithm in stacked auto-encoder training and to see how the proposed strategy works, the algorithm is tested on MINIST and Fashion-MINIST as Fig. 6. MINIST data set is first used to train and test deep neural networks, which contains 60,000 training images and 10,000 testing images of digital handwriting (0–9) with a size of 28*28. In order to evaluate the model adequately, Fashion-MNIST (Ledesma et al., 2019) is adopted, which is a fashion product images dataset as shown in Fig. 6, whilst providing a more challenging classification model. Some experiments (after section 4.3) are based on Fashion-MINIST.

The experiments are implemented on three levels. First, the computing environment is tested, especially the effects of parameter quantization. Second, the effectiveness of the improved estimation of the distribution algorithm is evaluated by different stacked auto-encoders and then a deeper stacked auto-encoder is adopted to testify the optimization capability of improved estimation of the distribution algorithm. In the experiments, the population size is set as 100. The architecture of the neural network is defined as 784-120-10, which contains 784 input neurons, 120 hidden neurons and 10 output neurons. Other architectures, such as 784-300-100-10 and 784-1000-500-250-30-10 are adopted. The following sections give more details about the experimental design and results.

5.1. The effects of parameter quantization

In order to accelerate the calculation process, NVIDIA GPU is adopted. However, the single-precision data is used in the GPU and the double-precision data is always used in MATLAB based on the CPU. Additionally, the output data precision of the probabilistic sampling function in MATLAB is limited with 4 decimal places resolution. Therefore, a comparison is carried out to exhibit the affection of data precision for the optimization on MNIST dataset. First, a traditional SGD algorithm is adopted to exhibit the effect of data precision, and the validation of pre-training is tested. The first architecture of the neural network is 784–120–10. The parameters with the double resolution are compared with the data with 4 decimal places of resolution as indicated in Fig. 8 which denotes that a better result is obtained when the parameters are with high precision. An additional test is carried out under the situation of with or without pre-training. According to Fig. 7, the neural network without pre-training has a worse result. The layer-by-layer pre-training can search a better region for the neural network parameters, and then the fine-tuning process will optimize the solutions accurately. Therefore, the pre-training is effective for the SGD algorithm.

5.2. Effectiveness of improved estimation of distribution algorithm & one-step pre-training

For traditional estimation of distribution algorithm, all the individuals are generated by the sampling of probabilistic model, which was constructed based on the statistical information of promising individuals. For large-scale optimization problems, the probability sampling will increase the calculation time and there will be a serious combination explosion problem. In the improved estimation of the distribution algorithm, the population is updated by two different strategies instead of single probability sampling. Namely, some individuals are generated by probability sampling and the rest of them are replaced by the mean of corresponding advanced individuals. The architecture of the neural network is also 784-120-10, and the pre-training procedure is also adopted. The number of parameters can be optimized with a size of 784*120 (weights between input and hidden layer) + 120*784 (weights

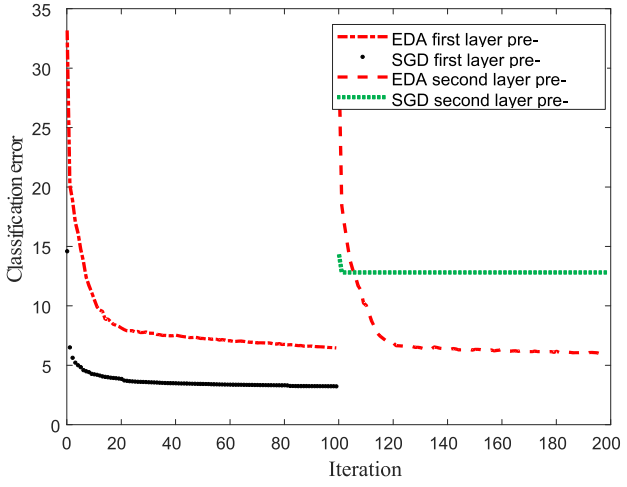


Fig. 9. Diagram of pre-training for SGD and EDA on MNIST.

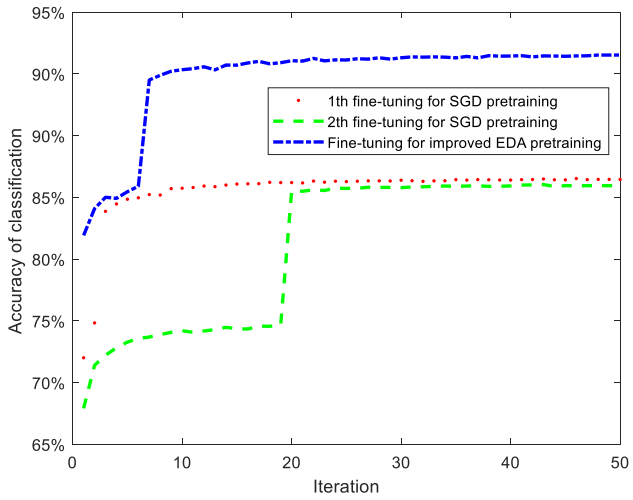


Fig. 10. Comparison of fine-tuning result for SGD and EDA pre-training on MNIST.

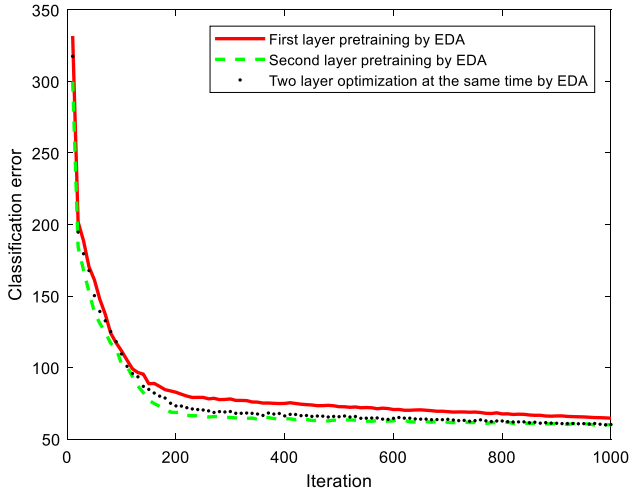


Fig. 11. Comparison of EDA pre-training and without pre-training on MNIST.

between the hidden layer and output layer) + 120(bias of hidden layer) + 784 (bias of output layer) = 189,064, which is a large-scale one for estimation of distribution algorithm. Therefore, it is convincing to describe the large-scale optimization ability of the improved algorithm. The comparison of the optimization result is shown in Fig. 8. The improved estimation of the distribution algorithm has a better performance than the traditional one.

The deeper neural network architecture is 784-300-100-10. The parameter of the deep neural network is also trained layer-by-layer with the estimation of the distribution algorithm and then a fine-tuning is carried out to search the accurate parameters by the SGD algorithm. According to the principle of pre-training, the first hidden layer is trained and then the second hidden layer is trained by the first hidden layer output. The loss value of pre-training by SGD and improved estimation of the distribution algorithm are shown in Fig. 10. The pre-training of two hidden layers has a similar convergent performance by improved estimation of the distribution algorithm. For the SGD algorithm, the first hidden layer has a better performance. However, the performance is worse in the second hidden layer pre-training as displayed in Fig. 9.

In order to analyze the layer-wise pre-training effectiveness, the second procedure of fine-tuning is carried out. The SGD based layer-wise pre-training has worse classification performance than the EDA

Table 2
No. of parameters in pre-training on MNIST.

Architectures	No. of parameters
784-120-10	189,064
784-300-100-10	344,784
784-1000-500-250-30-10	1,442,584

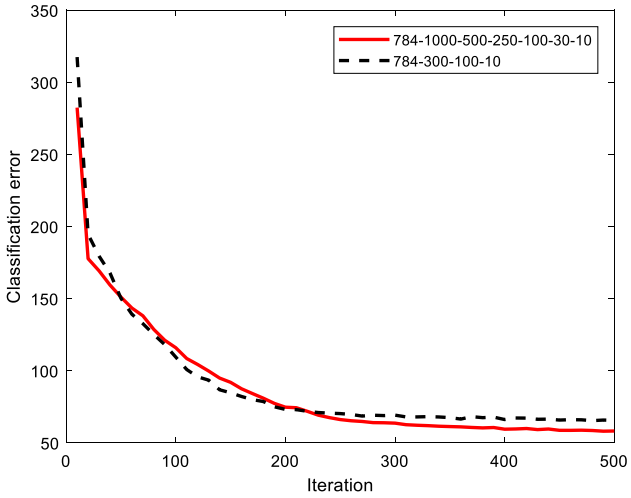


Fig. 12. Comparison of convergence error under different architectures on MNIST.

algorithm-based layer-wise pre-training as shown in Fig. 10, which looks like trapped into the local minima (twice repeated procedures are carried out for SGD algorithm).

In order to analyze the search ability of improved estimation of distribution algorithm further, the parameters of the whole neural network are optimized by improved estimation of distribution algorithm without layer-wise pre-training. The performance of the optimization is shown in Fig. 11, denoting that the improved estimation of the distribution algorithm has a similar performance for the whole neural network parameter optimization as the layer-wise pre-training. It means that the improved estimation of the distribution algorithm has a promising search efficiency and accuracy to search the whole neural network directly based on the improved estimation of the distribution algorithm. The layer-wise pre-training procedure can be replaced with the one-step pre-training.

A deeper architecture, which is a typical one, is adopted as a reference (Hinton and Salakhutdinov, 2006) to verify the search ability of the improved algorithm further. The architecture of the neural network is 784-1000-500-250-30-10. Table 2 shows the number of parameters for pre-training. It can be seen that there is a huge large-scale optimization problem. According to the convergent process of improved estimation of distribution algorithm (Fig. 12), the performance of improved estimation of distribution algorithm is stable for a deeper neural network. The convergent speed and classification error are also similar to the shallower one. Therefore, the depth of the neural network has limited influence on the convergence of the algorithm.

After the one-step pre-training, a fine-tuning process is carried out. The classification accuracy is about 99.9% for MNIST which has been reached to 100%, and there are no intervals for comparison. Therefore, the Fashion-MNIST dataset is adopted for the following verifications.

5.3. Effectiveness of one-step estimation of distribution algorithm pre-training & gradient fine-tuning

The local optimization capability of estimation of distribution algorithm is limited. And, conversely, the gradient-based algorithm has

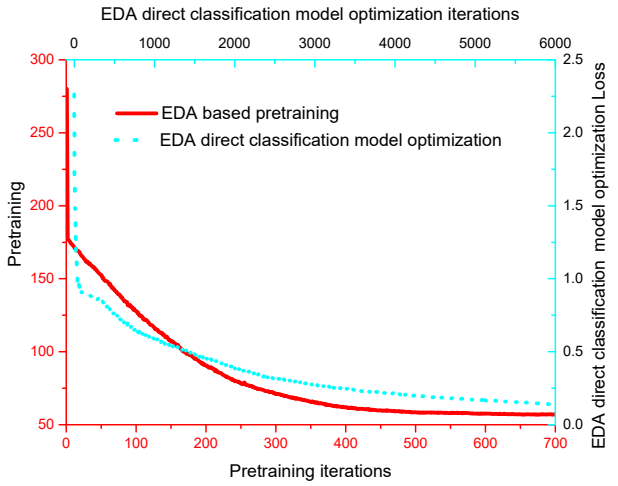


Fig. 13. The descent of loss for two models on Fashion-MNIST.

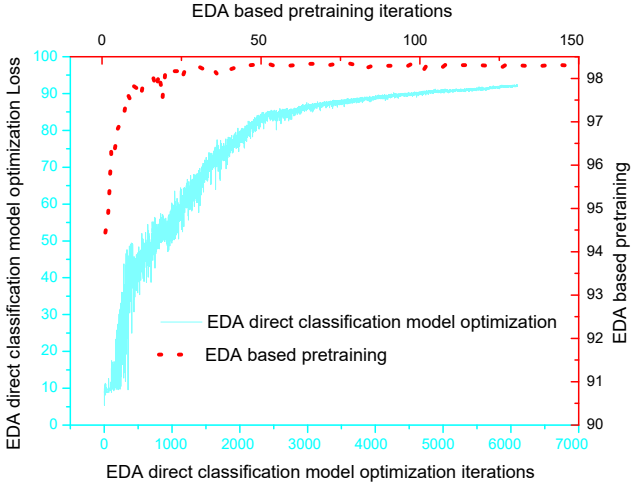


Fig. 14. The classification accuracy comparison of two models on Fashion-MNIST.

better local optimization capability. Therefore, estimation of distribution algorithm-based pre-training combination of the gradient-based fine-tuning procedure is adopted. In order to verify the effectiveness of this strategy, a comparing experiment is carried out. The baseline of the neural network is also 784-1000-500-250-30. The complete model in one-step pre-training is 784-1000-500-250-30-784, and then the model in fine-tuning is 784-1000-500-250-30-10. A direct classification model of 784-1000-500-250-30-10 is trained by the estimation of the distribution algorithm, which removes the pre-training process. Fig. 13 compares the loss for two algorithms. They have the same tendency of loss. The loss value is small in the direct classification model, which is the classification loss instead of reconstruction error in the pre-training process. Fig. 14 is the classification accuracy comparison. Although the classification accuracy goes up with the iterations as the gradient fine-tuning strategy (accuracy 92.36% & 98.24%), the direct classification optimization model has lower classification accuracy than using the pre-training way. In the two figures, the two models have a different x–y axis. Therefore, estimation of distribution algorithm-based one-step pre-training combines with gradient fine-tuning strategy is effective.

5.4. Different mask ratio and promising individual testifying

5.4.1. Model evaluation based on different mask ratio

The mask ratio (MR) affects the population updating strategy. In

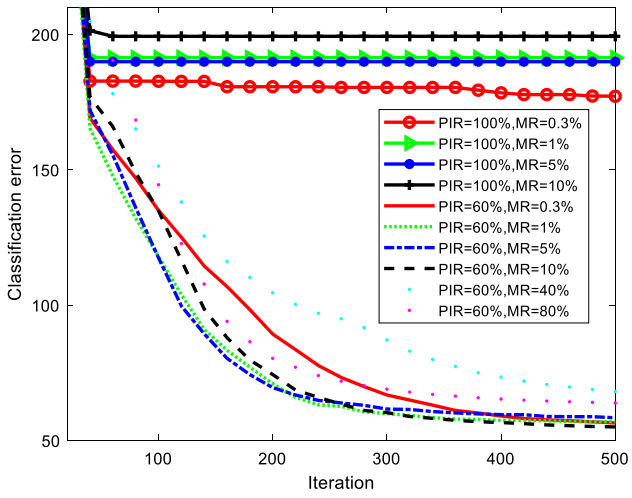


Fig. 15. The pre-training process under different MR on Fashion-MNIST.

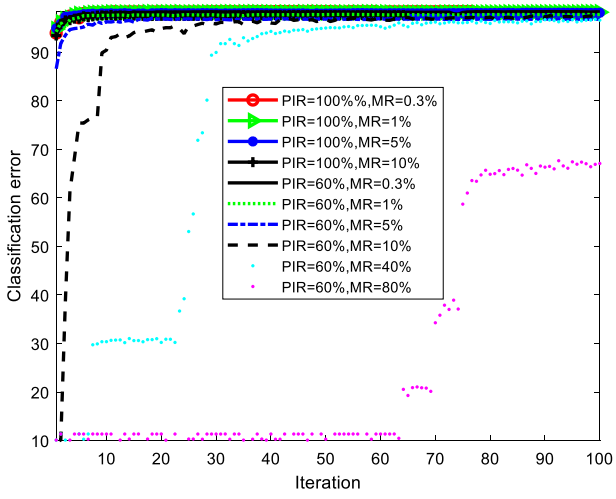


Fig. 16. The fine-tuning process under different MR on Fashion-MNIST.

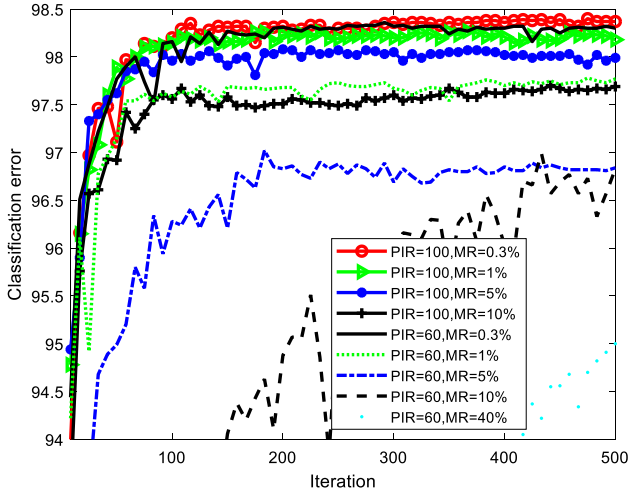


Fig. 17. Amplification of local region of Fig. 16.

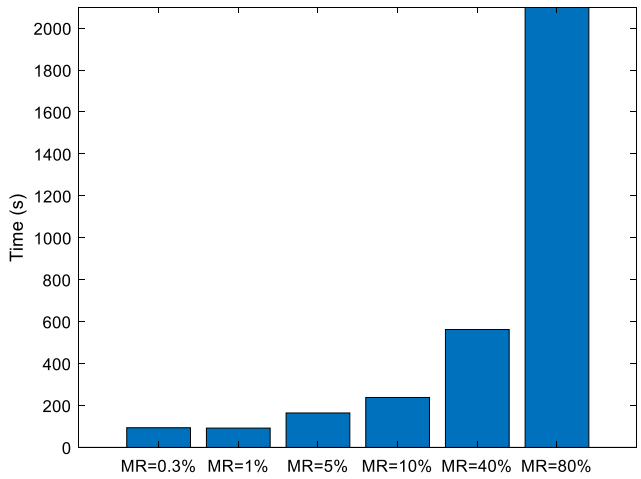


Fig. 18. Time consumption under PIR = 60 on Fashion-MNIST.

order to analyze the influence of different mask ratios on the performance of estimation of distribution algorithm, experiments are carried out. Some typical mask ratios, such as 0.3%, 1%, 5%, 10% and 40% are adopted for updating the population when the promising individual ratio (PIR) is 60% and 100%. The promising individual ratio will affect μ and σ of probabilistic model. The convergent performance under PIR = 100 and 60 are different in the pre-training as shown in Fig. 15. A bigger promising individual ratio slows down the convergence speed of the algorithm. Therefore, the convergent speed under PIR = 100% is worse than PIR = 60%. For PIR = 60%, the convergent performance is promising. As a whole, a bigger MR goes against the convergent performance. The fine-tuning process is shown in Fig. 16 and Fig. 17. Fig. 17 is the amplification of the local region given in Fig. 16. According to the two figures, the pre-training is valid under smaller MR (Fig. 16). A larger MR increases the probability of combination explosion. Therefore, the pre-training is invalid when MR is 10%, 40% and 80% as Fig. 17, which means the parameters gathered by pre-training are the same as random values. According to Fig. 18, the fine-tuning results are similar under the parameters of (PIR = 100, MR = 0.3%), (PIR = 100, MR = 1%) and (PIR = 60, MR = 0.3%). Considering the efficiency of the algorithm (a bigger PIR and MR will consume much time as Fig. 19), the parameter of (PIR = 60, MR = 0.3%) is a better choice.

According to Fig. 17, it is interesting to note that the convergent performance of pre-training is worse for PIR = 100. Nevertheless, the pre-training effect is promising in case of the satisfactory initial value. Fig. 19 may explain this vividly. The estimation of the distribution algorithm based on pre-training is shown in Fig. 19(a). Supposing estimation of distribution algorithm has found the region ③ instead of local minimal region ① and ②. However, the probabilistic model under PIR = 100 has a bigger σ as Fig. 19(a), the randomness of probabilistic sampling will be stronger. Although the estimation of the distribution algorithm has found region ③, it isn't easy to get the valley point. Conversely, SGD has better local optimization ability than the estimation of the distribution algorithm. Therefore, it can realize the accurate solution searching under region ③ which was found by estimation of distribution algorithm.

5.4.2. Model evaluation based on the different promising individual ratios

The promising individual ratio (PIR) will affect the construction of the probabilistic model. A smaller promising individual ratio will increase the convergence speed. However, it leads to the early maturing of the algorithm. A bigger promising individual ratio slows down the convergence speed of the algorithm. A different promising individual ratio is adopted for testing the influence of promising individuals when MR = 0.3% and 5%. Fig. 20 shows the estimation of distribution algorithm-based pre-training loss. We can see that a smaller promising individual ratio accelerates the convergence speed of the algorithm. The convergence speed is very slow than others when PIR = 100%. However, in the fine-tuning process diagram as Fig. 21, the pre-training is invalid when the PIR is small (MR = 5%, PIR = 10% or MR = 5%, PIR = 40%), even though the pre-training process is promising. Although, the pre-training process is bad when PIR = 100%, the fine-tuning process is promising as Fig. 22. According to Figure 23, PIR = 60%, MR = 0.3% are the better parameters for comprehensive consideration.

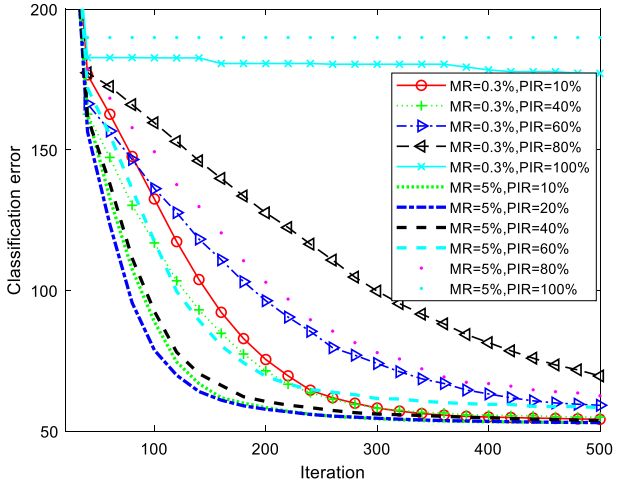
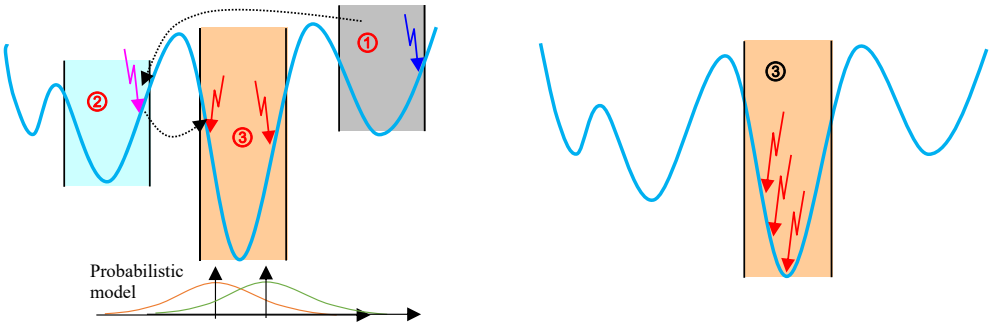


Fig. 20. The pre-training process under different PIR on Fashion-MNIST.



(a) EDA Pre-training (b) SGD Fine-tuning

Fig. 19. Pre-training and fine-tuning based optimization diagram.

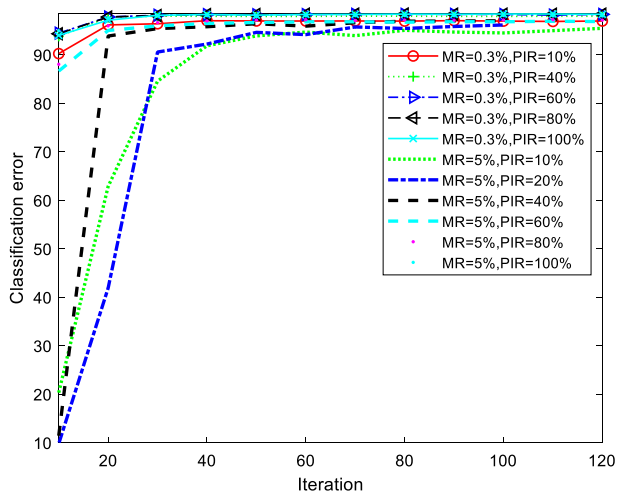


Fig. 21. The fine-tuning process under different PIR on Fashion-MNIST.

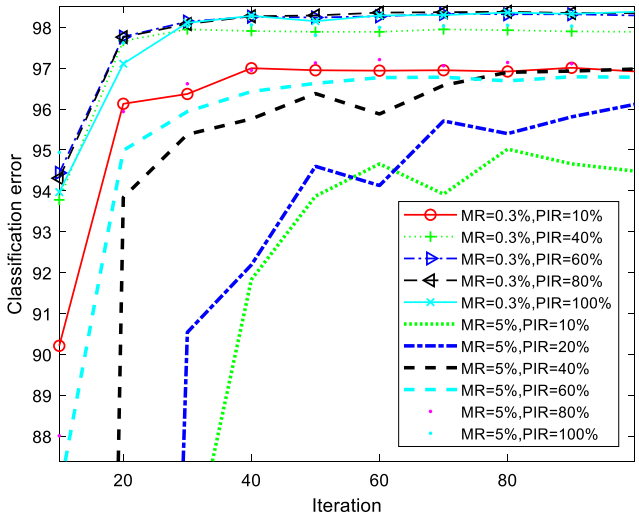


Fig. 22. The amplification of the local region of Fig. 21.

5.5. Compared with state-of-the-art

Fashion-MNIST dataset is a challenging benchmark for classification models. The stacked auto-encoder with the architecture of 784-1000-500-250-30-10 is also used for classification, PIR = 60% and MR = 0.3%. The same procedures, one-step pre-training and fine-tuning, are carried out. The proposed model is compared with state-of-the-art models on Fashion-MNIST as Table 3. Compared with traditional MINIST, the classification accuracy is lower, and our model can achieve a result of 98.38%. The references with the mark [*] come from the automatic benchmarking system for Fashion-MNIST, which covers various kinds of classifiers based on the convolution neural network. This network is popular in image processing. Therefore, the studies are always based on the convolution neural network, such as evoCNN (Xu et al., 2019) is based on the evolutionary algorithm, psoCNN (Junior and Yen, 2019) adopts particle swarm optimization algorithm, etc. However, it is not possible to make a direct comparison with our algorithm due to the different structures. Evolving unsupervised deep neural network (EUDNN) (Sun et al., 2019) is the only study for deep neural network optimization in which direct comparison can be made. The neural network structure of EUDNN (784-400-202-106-88-10) is adopted for testifying. The proposed algorithm gets a similar accuracy (98.92) as EUDNN. However, the SGD-based model has a lower accuracy (95.74%), which means that the heuristic algorithm has played an active role in the weights' optimization. Due to the complexity of Fashion-MNIST, the structure of 784-400-202-106-88-10 is not enough for classification, even though, the performance of the algorithm is better than SGD based algorithm.

Table 3
Comparison of state-of-the-arts.

Models	Dataset	Accuracy (%)
5 Conv + BN + pooling[*]	Fashion-MNIST	63.1
MobileNet[*]	Fashion-MNIST	95.0
DenseNet-BC[*]	Fashion-MNIST	95.4
Dual path network with wide resnet 28–10[*]	Fashion-MNIST	95.7
WRN-28-10 + Random Erasing[*]	Fashion-MNIST	96.3
Google AutoML[*]	Fashion-MNIST	93.9
evoCNN (Xu et al., 2019)	Fashion-MNIST	94.53
psoCNN + dropout + BN (best) (Junior and Yen, 2019)	Fashion-MNIST	94.47
Hermite Convolutional Networks (Sabour et al., 2017)	Fashion-MNIST	97.19
Adaptive Learning Rate CNN (Lopez-Rincon et al., 2018)	Fashion-MNIST	92.01
Adaptive Learning Rate ResNet (Lopez-Rincon et al., 2018)	Fashion-MNIST	91.92
CapsNet (Zhang et al., 2019)	Fashion-MNIST	89.71
DCaps (Chollet, 2017)	Fashion-MNIST	91.18
Xception [71]	Fashion-MNIST	92.45
ResNet50 (Chollet, 2017)	Fashion-MNIST	92.03
Our algorithm (784-1000-500-250-30–10)	Fashion-MNIST	98.38
EUDNN (Sun et al., 2019) (784-400-202–106-88–10)	MNIST	98.85
SGD based (784-400-202-106-88-10)	MNIST	95.74
Our algorithm (784-400-202-106-88–10)	MNIST	98.92
SGD based (784-400-202-106-88-10)	Fashion-MNIST	88.49
Our algorithm (784-400-202-106-88–10)	Fashion-MNIST	90.13

[*]The dataset is available at https://github.com/zalandoresearch/fashion-mnist.

6. Conclusion

As a heuristic algorithm, the estimation of the distribution algorithm has been proposed for different kinds of applications. However, the capability of large-scale optimization is limited due to the curse of dimensionality. The weight training of deep neural networks is a large-scale optimization problem. Although the traditional gradient-based algorithm can achieve this goal, some inherent weaknesses are leading to failure in training, such as gradient disappearing. The heuristic algorithm doesn't make use of a gradient to evaluate and update individuals. Therefore, a new attempt is proposed to optimize the weight of the deep neural network based on an improved estimation of the distribution algorithm. This paper explores the capability of the improved EDA and demonstrates that it is effective for deep neural network parameter searching and time reduction. In addition, it is a universal framework, which can be extended to other deep neural networks. However, the optimization is only for weights, and the architecture is exclusive. This will be our future work.

CRediT authorship contribution statement

Qingyang Xu: Conceptualization, Methodology, Writing - review & editing. **Anbang Liu:** Investigation, Software. **Xianfeng Yuan:** Formal analysis. **Yong Song:** Funding acquisition. **Chengjin Zhang:** Formal analysis. **Yibin Li:** Project administration.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported by the National Key Research and Development Plan of China under Grant (2017YFB1300205, 2020AAA0108903), National Natural Science Foundation of China under Grants (61803227, 61573213, 61603214, 61673245), Natural Science Foundation of Shandong Province (ZR2020MD041, ZR2020MF077).

References

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521*, 436–444.
Kuremoto, T., Kimura, S., Kobayashi, K., & Obayashi, M. (2014). Time series forecasting using a deep belief network with restricted Boltzmann machines. *Neurocomputing, 137*, 47–56.
Wang, N., Er, M. J., & Han, M. (2017). Generalized single-hidden layer feedforward networks for regression problems. *IEEE Transactions on Neural Networks & Learning Systems, 26*, 1161–1176.
Martín, A., Lara-Cabrera, R., Fuentes-Hurtado, F., Naranjo, V., & Camacho, D. (2018). EvoDeep: A new evolutionary approach for automatic Deep Neural Networks parametrisation. *Journal of Parallel and Distributed Computing, 117*, 180–191.
Ye, F. (2017). Particle swarm optimization-based automatic parameter selection for deep neural networks and its applications in large-scale and high-dimensional data. *PLoS ONE, 12*, Article e0188746.
Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science, 313*, 504–507.
Hinz, T., Navarroguerrero, N., Magg, S., & Wermter, S. (2018). Speeding up the Hyperparameter Optimization of Deep Convolutional Neural Networks. *International Journal of Computational Intelligence & Applications, 17*, 1850008.
Assunção, F., Lourenço, N., Machado, P., & Ribeiro, B. (2019). DENSER: Deep evolutionary network structured representation. *Genetic Programming and Evolvable Machines, 20*, 5–35.
Junior, F. E. F., & Yen, G. G. (2019). Particle swarm optimization of deep neural networks architectures for image classification. *Swarm and Evolutionary Computation, 49*, 62–74.
Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *The Journal of Machine Learning Research, 13*, 281–305.
Sun, Y., Yen, G. G., & Yi, Z. (2019). Evolving Unsupervised Deep Neural Networks for Learning Meaningful Representations. *IEEE Transactions on Evolutionary Computation, 23*, 89–103.
Zhang, Y., Cheng, S., Shi, Y., Gong, D., & Zhao, X. (2019). Cost-sensitive feature selection using two-archive multi-objective artificial bee colony algorithm. *Expert Systems with Applications, 137*, 46–58.
, 2020K. Li, T. Zhang, R. Wang, Deep Reinforcement Learning for Multi-objective Optimization, IEEE T CYBERNETICS, (2020) In Press.
Ma, L., Cheng, S., & Shi, Y. (2020). Enhancing Learning Efficiency of Brain Storm Optimization via Orthogonal Learning Design. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 1–20.
Guo, Y., Zhang, X., Gong, D., Zhang, Z., & Yang, J. (2020). Novel Interactive Preference-Based Multiobjective Evolutionary Optimization for Bolt Supporting Networks. *IEEE Transactions on Evolutionary Computation, 24*, 750–764.
Oong, T. H., & Isa, N. A. M. (2011). Adaptive Evolutionary Artificial Neural Networks for Pattern Classification. *IEEE Transactions on Neural Networks, 22*, 1823–1836.
Al-Dabbagh, R. D., Mekhilef, S., & Baba, M. S. (2015). Parameters' fine tuning of differential evolution algorithm. *The Computer Systems Science and Engineering, 30*, 125–139.
Al-Dabbagh, M. D., Al-Dabbagh, R. D., Abdullah, R. S. A. R., & Hashim, F. (2015). A new modified differential evolution algorithm scheme-based linear frequency modulation radar signal de-noising. *Optimization and Engineering, 47*, 771–787.
Yao, X., & Liu, Y. (1997). A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks, 8*, 694–713.
Hussain, K., Salleh, M. N. M., Cheng, S., & Shi, Y. (2019). Metaheuristic research: A comprehensive survey. *Artificial Intelligence Review, 52*, 2191–2233.
Lv, B., Yang, B., Zhu, X., & Li, J. (2019). Operational optimization of transit consolidation in multimodal transport. *Computers and Industrial Engineering, 129*, 454–464.
Hu, B., & Yang, B. (2019). A particle swarm optimization algorithm for multi-row facility layout problem in semiconductor fabrication. *Journal of Ambient Intelligence and Humanized Computing, 10*, 3201–3210.
Wu, D., Liao, Y., Hu, C., Yu, S., Tian, Q. (2020). An Enhanced Fuzzy Control Strategy for Low-Level Thrusters in Marine Dynamic Positioning Systems Based on Chaotic Random Distribution Harmony Search. The International Journal of Fuzzy Systems.
Wu, D., Ren, F., & Zhang, W. (2016). An energy optimal thrust allocation method for the marine dynamic positioning system based on adaptive hybrid artificial bee colony algorithm. *Ocean Engineering, 118*, 216–226.
Wu, D., Liu, X., Ren, F., & Yin, Z. (2016). An Improved Thrust Allocation Method for Marine Dynamic Positioning System. *Naval Engineers Journal, 129*, 89–98.
Wu, D., Ren, F., Qiao, L., & Zhang, W. (2018). Active disturbance rejection controller design for dynamically positioned vessels based on adaptive hybrid biogeography-based optimization and differential evolution. *ISA T, 78*, 56–65.
Cheng, S., Ma, L., Lu, H., Lei, X., Shi, Y. (2020). Evolutionary computation for solving search-based data analytics problems. Artificial Intelligence Review.
Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation, 10*, 99–127.
Snoek, J., Larochelle, H., Adams, R. P. (2012). Practical Bayesian Optimization of Machine Learning Algorithms, Proceedings of the 25th International Conference on Neural Information Processing Systems, USA, pp. 2951–2959.
Bello, I., Zoph, B., Vasudevan, V., & Le, Q. V. (2017). *Neural Optimizer Search with Reinforcement Learning, Proceedings of Machine Learning Research* (pp. 459–468). Sydney, Australia: International Convention Centre.
Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 8697–8710).
Zhong, Z., Yan, J., Wu, W., Shao, J., Liu, C. (2018). Practical block-wise neural network architecture generation, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, USA, pp. 2423–2432.
Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Tan, J., Le, Q.V., Kurakin, A. (2017). Large-Scale Evolution of Image Classifiers, Proceedings of Machine Learning Research, Sydney, Australia, 2017, pp. 2902–2911.
Xie, S., Zheng, H., Liu, C., Lin, L., SNAS: stochastic neural architecture search, Proceedings of the International Conference on Learning Representations, New Orleans, Louisiana, USA, 2018.
Cai, H., Zhu, L., Han, S. (2018). Proxylessnas: Direct neural architecture search on target task and hardware, Proceedings of the International Conference on Learning Representations, New Orleans, Louisiana, USA, 2018.
Zhang, C., Ren, M., Urtasun, R. (2018). Graph hypernetworks for neural architecture search, Proceedings of the International Conference on Learning Representations, New Orleans, Louisiana, USA, 2018.
Sun, Y., Xue, B., Zhang, M., & Yen, G. G. (2020). Evolving Deep Convolutional Neural Networks for Image Classification. *IEEE Transactions on Evolutionary Computation, 24*, 394–407.
Xu, Z., Dai, L., Kemp, A.M., Metz, J. (2019). Learning an Adaptive Learning Rate Schedule, arXiv preprint, (2019) arXiv:1909.09712.
Lopez-Rincon, A., Tonda, A., Elati, M., Schwander, O., Piwowarski, B., & Gallinari, P. (2018). Evolutionary optimization of convolutional neural networks for cancer miRNA biomarkers classification. *Applied Soft Computing, 65*, 91–100.
Baldominos, A., Saez, Y., & Isasi, P. (2018). Evolutionary convolutional neural networks: An application to handwriting recognition. *Neurocomputing, 283*, 38–52.
Miuhlenbein, H., & Paaß, G. (1996). In *From Recombination of Genes to the Estimation of Distributions I. Binary Parameters* (pp. 178–187). London, UK: Springer-Verlag.
Dong, W., Chen, T., Tiňo, P., & Yao, X. (2013). Scaling Up Estimation of Distribution Algorithms for Continuous Optimization. *IEEE Transactions on Evolutionary Computation, 17*, 797–822.
Sun, B. Q., Wang, L., & Peng, Z. P. (2020). Bound-guided hybrid estimation of distribution algorithm for energy-efficient robotic assembly line balancing. *Computers & Industrial Engineering, 146*.
Wang, H., Chien, C., & Gen, M. (2015). An Algorithm of Multi-Subpopulation Parameters With Hybrid Estimation of Distribution for Semiconductor Scheduling With Constrained Waiting Time. *IEEE Transactions on Semiconductor Manufacturing, 28*, 353–366.
Pérez-Rodríguez, R., & Hernández-Aguirre, A. (2019). A hybrid estimation of distribution algorithm for the vehicle routing problem with time windows. *Computers & Industrial Engineering, 130*, 75–96.
Arin, A., & Rabadi, G. (2017). Integrating estimation of distribution algorithms versus Q-learning into Meta-RaPS for solving the 0–1 multidimensional knapsack problem. *Computers & Industrial Engineering, 112*, 706–720.
Wang, L., Wang, S., Xu, Y., Zhou, G., & Liu, M. (2012). A bi-population based estimation of distribution algorithm for the flexible job-shop scheduling problem. *Computers & Industrial Engineering, 62*, 917–926.
Chen, T., Tang, K., Chen, G., & Yao, X. (2010). Analysis of Computational Time of Simple Estimation of Distribution Algorithms. *IEEE T EVOLUT COMPUT, 14*, 1–22.
Mishra, K. M., & Gallagher, M. (2014). *A Modified Screening Estimation of Distribution Algorithm for Large-Scale Continuous Optimization, Asia-Pacific Conference on* (pp. 119–130). Dunedin, New Zealand: Simulated Evolution and Learning.
Hansen, N., & Ostermeier, A. (2001). Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation, 9*, 159–195.
Wang, Y., Li, B. (2008). A restart univariate estimation of distribution algorithm: sampling under mixed Gaussian and Lévy probability distribution, 2008 IEEE Congress on Evolutionary Computation, Hong Kong, China, 2008, pp. 3917-3924.
Bielza, C., Robles, V., & Larranaga, P. (2009). Estimation of distribution algorithms as logistic regression regularizers of microarray classifiers. *Methods of Information in Medicine, 48*, 236–241.

Karshenas, H., Santana, R., Bielza, C., & Larrañaga, P. (2013). Regularized continuous estimation of distribution algorithms. *Applied Soft Computing Journal, 13*, 2412–2432.

Bosman, P.A.N. (2009). On Empirical Memory Design, Faster Selection of Bayesian Factorizations and Parameter-free Gaussian EDAs, Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, New York, NY, USA, 2009, pp. 389-396.

Kabán, A., Bootkrajang, J., & Durrant, R. J. (2016). Toward Large-Scale Continuous EDA: A Random Matrix Theory Perspective. *Evolutionary Computation, 24*, 255–291.

Omidvar, M. N., Yang, M., Mei, Y., Li, X., & Yao, X. (2017). DG2: A Faster and More Accurate Differential Grouping for Large-Scale Black-Box Optimization. *IEEE Transactions on Evolutionary Computation, 21*, 929–942.

Hauschild, M., & Pelikan, M. (2011). An introduction and survey of estimation of distribution algorithms. *Swarm and Evolutionary Computation, 1*, 111–128.

Li, H., Hong, Y., Kwong, S., Ren, Q., & Wang, W. (2012). Enhancement of continuous estimation of distribution algorithms by density ensembles. *Engineering Optimization, 44*, 1303–1320.

Ahmed, A., Khan, Q., Naeem, M., Iqbal, M., Anpalagan, A., & Awais, M. (2018). An insight to the performance of estimation of distribution algorithm for multiple line outage identification. *Swarm and Evolutionary Computation, 39*, 114–122.

Lu, H., Zhou, R., Cheng, S., & Shi, Y. (2019). Multi-center variable-scale search algorithm for combinatorial optimization problems with the multimodal property. *Applied Soft Computing, 84*, Article 105726.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research, 15*, 1929–1958.

Ahn, C. W., & Ramakrishna, R. S. (2003). Elitism-based compact genetic algorithms. *IEEE Transactions on Evolutionary Computation, 7*, 367–385.

Purshouse, R. C., Fleming, P. J. (2002). Why use elitism and sharing in a multi-objective genetic algorithm, Proceedings of the Genetic and Evolutionary Computation Conference, New York, USA, 2002, pp. 520–527.

Gao, S., & de Silva, C. W. (2018). Estimation distribution algorithms on constrained optimization problems. *Applied Mathematics and Computation, 339*, 323–345.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature, 323*, 533–536.

Xiao, H., Rasul, K., Vollgraf, R. (2017). Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms, arXiv, (2017) arXiv/1708.07747.

Ledesma, L., Olveres, J., Escalante-Ram I Rez, B. (2019). Hermite Convolutional Networks, Iberoamerican Congress on Pattern Recognition, Springer, Havana, Cuba, 2019, pp. 398-407.

Sabour, S., Frosst, N., & Hinton, G. E. (2017). *Dynamic Routing Between Capsules, Advances in Neural Information Processing Systems 30* (pp. 3856–3866). Curran Associates: Inc.

Zhang, X., Sun, Y., Wang, Y., Li, Z., Li, N., & Su, J. (2019). A novel effective and efficient capsule network via bottleneck residual block and automated gradual pruning. *Computers & Electrical Engineering, 80*, Article 106481.

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions, Proceedings of the IEEE conference on computer vision and pattern recognition, Honolulu, Hawai, USA, 2017, pp. 1251–1258.