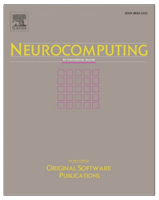
Neurocomputing 442 (2021) 161–172

Contents lists available at ScienceDirect

Neurocomputing

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



HELP: An LSTM-based approach to hyperparameter exploration in neural network learning

Wendi Li [a], Wing W. Y. Ng [a,*], Ting Wang [a,*], Marcello Pelillo [b], Sam Kwong [c]

[a] Guangdong Provincial Key Laboratory of Computational Intelligence and Cyberspace Information, Computer Science and Engineering, South China University of Technology, Guangzhou, China
[b] Department of Environmental Sciences, Informatics and Statistics and with the European Centre for Living Technology, University of Venice, Italy
[c] Department of Computer Science, City University of Hong Kong, Hong Kong, China

A R T I C L E I N F O

Article history:
Received 18 February 2020
Revised 21 December 2020
Accepted 29 December 2020
Available online 4 March 2021
Communicated by Yue Wu

Keywords:
Hyperparameter tuning
Deep neural network
Generative adversarial net
Convolutional neural network

A B S T R A C T

Hyperparameter selection is very important for the success of deep neural network training. Random search of hyperparameters for deep neural networks may take a long time to converge and yield good results because the training of deep neural networks with a huge number of parameters for every selected hyperparameter is very time-consuming. In this work, we propose the Hyperparameter Exploration LSTM-Predictor (HELP) which is an improved random exploring method using a probability-based exploration with an LSTM-based prediction. The HELP has a higher probability to find a better hyperparameter with less time. The HELP uses a series of hyperparameters in a time period as input and predicts the fitness values of these hyperparameters. Then, exploration directions in the hyperparameter space yielding higher fitness values will have higher probabilities to be explored in the next turn. Experimental results for training both the Generative Adversarial Net and the Convolution Neural Network show that the HELP finds hyperparameters yielding better results and converges faster.

1. Introduction

Neural networks have been successfully applied in many areas [1,2]. However, the success of a particular neural network depends upon the selection of hyperparameters to yield optimal performance [3]. For instance, hyperparameters include the optimization and tuning of the model structure, the step size of a gradient-based optimization, and the data presentation which significantly impacts the learning process and the network performance. In contrast to the great success in practical problems, the selection and the optimization of hyperparameters are still open research problems for deep neural network learning [4].

The most widely used method to select hyperparameters is either traditional random search [5] or manual selection based on user experience. However, these ad hoc selections may require a large number of trial-and-error steps to achieve satisfactory results. Evolution-based strategies, such as Population Based Training (PBT) [6], Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [7] and genetic algorithms [8], can be split into two parts: exploring new genotypes randomly and exploiting useful genotypes to form more individuals. PBT and CMA-ES perform explorations around good hyperparameters while genetic algorithms perform explorations by applying the mutation operation on the previous generation's individuals. Such explorations are random because both the mutation and the perturbation do not use any learned knowledge and attempt to try new genotypes around existing ones blindly. The major drawback of the random exploration method is that it does not use any information from previous iterations because iterations are independent [9].

To enhance the efficiency of exploration steps, we apply the Long Short-Term Memory (LSTM) [10] to assign a probability to each exploration direction in the hyperparameter space based on its predictions. The prediction is made based on both currently and previously trained neural networks.

The major contributions of this work include:

1. Our paper proposes the Hyperparameter Exploration LSTM-Predictor (HELP) structure which guides the hyperparameter tuning process of the neural network training. The HELP is a general method that can be applied to all kinds of hyperparameter tuning methods with random exploring process. As a result, it can significantly improve the efficiency of the exploration process.

* Corresponding authors.
E-mail addresses: wingng@ieee.org (W.W. Y. Ng), tingwang@ieee.org (T. Wang).

https://doi.org/10.1016/j.neucom.2020.12.133
0925-2312/© 2021 Elsevier B.V. All rights reserved.

2. The HELP is based on a Recurrent Neural Network (RNN) that learns the hyperparameter and its effect in iterations of the neural network training process. Compared with the typical hyperparameter tuning strategies, the major improvement of the HELP is that it not only retains the successful historical examples but also learns from failure lessons. Thus, not only the HELP has a higher probability of finding more efficient hyperparameters, but also it avoids bad hyperparameters in future exploration.
3. Different from typical optimization problems, the optimal hyperparameters are not fixed in for all iterations because deep neural networks need different hyperparameters in different training stages. The HELP collects the information during the training process in order to predict and find the optimal hyperparameters for the objective function at the next iteration.

This paper is organized as follows. Section 2 reviews related works and Section 3 introduces LSTM networks. Section 4 proposes the HELP algorithm. Then, experimental results and discussions are presented in Section 5. Finally, Section 6 concludes this work.

2. Related works

Major hyperparameter tuning methods for deep neural networks are traditional random search [5,11] and manual selection [12]. Traditional random search simply selects a set of random hyperparameters in the hyperparameter space and trains multiple deep neural network threads with these hyperparameters. Finally, the trained neural network yielding the best performance is selected. Manual selection selects the hyperparameter based on empirical consideration or user experience. These methods are common and simple. However, these methods do not have the generalizability to be applied to various neural networks. With the neural network surpassed 100 layers like Highway Network [13] and Residual Network [14], it stands for that neural network will develop to larger and deeper network. This trend indicates when training a large deep neural network with millions of parameters, the training process can be really time-consuming. It may waste a lot of time to use this kind of inefficient hyperparameter tuning method, which reveals the shortcomings of traditional random search and manual selection.

Therefore, some methods aimed at effective and automatic hyperparameter tuning were proposed. Bayesian optimization is a kind of important hyperparameter tuning method. Spearmint [15] considers the variable cost of the learning experiment. Tree-structured Parzen estimator [16] applies modeling strategy and expected improvement for sequential model-based optimization. Sequential model-based algorithm configurations [17,18] introduce a more intricate instantiation of the sequential model-based optimization framework. These methods estimate the next set of hyperparameters by updating the posterior area of the hyperparameter space. However, the training process of the above methods is sequential, which means every new training course requires the full training process of the previous one. These methods are unsuitable for large and complicated deep neural network structures.

Thus, the parallel Bayesian optimization [19–22] methods are proposed to speed up the overall optimization process. However, although the Bayesian methods are suitable for optimizing model design hyperparameters, which is a set of fixed hyperparameters of the deep neural network during the training process. It is not suitable for Bayesian methods to optimize dynamic hyperparameters that change at each training iteration. Bayesian optimization is to find the beneficial value of an unknown function through repeated sampling successively. With the deep neural network evolving in each training iteration, its optimal hyperparameters change as the training phase changes, and the previous sampling results will be unavailable. Besides, the parallel Bayesian optimization is under the assumption of parallel detection points have the same objective function of hyperparameter fitness. In fact, the objective function of the deep neural network trained by different hyperparameters is not the same. More suitable hyperparameter tuning methods for dynamic hyperparameters during the training process are needed.

Therefore, the evolutionary strategy methods are used for hyperparameter tuning. Evolutionary strategy methods provide a set of candidate solutions for a problem whose evaluation is based on an objective function. The goal of the evolutionary strategy is to maximize the evaluation function based on the last generation and its performance.

Particle Swarm Optimization (PSO) [23] algorithm is applied to hyperparameter tuning in deep neural network [24] to achieve considerable results. Another approach [25] tries to combine PSO with a steepest gradient descent algorithm. CMA-ES [7] improves the exploration efficiency because of its variable standard deviation noise in the exploration space. These guided search optimization algorithms can accelerate the optimization of hyperparameter by dynamically adjusting the hyperparameter during the training process.

In hyperparameter tuning, the performance of the neural network cannot be directly calculated, it can be regarded as a black-box optimization problem. Solving the hyperparameter tuning problem by evolutionary algorithm [8], optimizing hyperparameter by genetic algorithm [26], or by CMA-ES [7] all present the feasibility of the evolutionary strategy in this kind of black-box optimization problems. The fitness value reflects the performance of a neural network. Hyperparameters for each trained neural network thread will be generated by the last hyperparameter generation and its performance.

Neural network hyperparameter tuning with the evolutionary strategy is a type of parallel search. This kind of method employs multiple threads to explore different positions in the hyperparameter space and try to rearrange more threads to the positions that have been shown excellent-result.

PBT [6] is a representative method in evolutionary strategy. It has two main processes: exploring new hyperparameters and rearranging threads to the currently discovered available positions in the hyperparameter space. These two processes are described as *explore* step and *exploit* step. In the *explore* step, threads are allowed to search new hyperparameters around their current positions. In the *exploit* step, the whole population will be sorted by a reliable estimation mechanism, for example, the accuracy rate on the validation set in classification problems. Then the latter threads will be replaced by the front threads by copying their current training weights and hyperparameters. The main idea of other evolutionary strategies [7,8,27–30] for hyperparameter tuning is similar to rearranging more neural networks threads near the threads that have good performance. Good threads also explore other positions near their original points in the hyperparameter space. They try to find whether the current position is optimal in their local region. Above these methods, PBT has shown the ability to yield excellent results due to its alternant process in rearranging threads and exploring new hyperparameter points. The alternation of *exploit* and *explore* step makes the whole population quickly drop the threads with bad performance, and allocate maximum computing resources for the excellent-result threads to explore new hyperparameter.

The evolutionary strategy in hyperparameter tuning has shown the advantages of dropping poor performers and exploring around good performers. But the exploring process in this kind of method is still inefficient [31]. All these methods still use random exploration with no information about the neural network training process considered [9]. Such behavior leads to an inefficient exploration process [31]. All the evolutionary strategies can be defined as three steps: generation, mutation, and fitness [32]. Mutation in evolution is nondirectional. Exploring the new hyperparameters using evolutionary strategies is completely random, which causes a waste of computing resources. Virtually, it usually has a higher probability to get a worse result by equiprobable exploratory direction choosing. Overlapping the explored hyperparameter during the nondirectional hyperparameter searching is also a problem that cannot be ignored. So our HELP method provides a model-agnostic approach that can be applied to evolutionary strategy methods. The proposed HELP is a predicting structure that uses RNN to learn previous information from neural networks' training process. To give effective guidance of the next step of the hyperparameter exploration, the HELP makes full use of all the information during the training process, including all the historical hyperparameter solutions and their performances. It will guide the hyperparameter tuning framework to explore by giving unequal probability through learned experience in the whole hyperparameter exploration space. As a result, the HELP provides higher efficiency of the exploration process and increases the probability of finding a better solution in a shorter time.

3. LSTM networks

RNN is aimed at processing time series data. The output at each moment depends on the previous sequence of inputs and the input at the current moment, because there is an internal state in RNN. Given an input sequence $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_T)$, RNN will calculate the corresponding internal state which is also called the hidden vector sequence $\hat{\boldsymbol{H}} = (\boldsymbol{\eta}_1, \boldsymbol{\eta}_2, \ldots, \boldsymbol{\eta}_T)$, and outputs the output sequence $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_T)$. At a given time t, current $\boldsymbol{\eta}_t$ and output $\boldsymbol{y}_t$ are calculated by the following:

$$\begin{aligned} \boldsymbol{\eta}_t &= \phi(\boldsymbol{x}_t \boldsymbol{W}_{x\eta} + \boldsymbol{\eta}_{t-1} \boldsymbol{W}_{\eta\eta} + \boldsymbol{b}_\eta) \\ \boldsymbol{y}_t &= \boldsymbol{\eta}_t \boldsymbol{W}_{\eta y} + \boldsymbol{b}_y \end{aligned}$$

where $\boldsymbol{W}$ represents the weight matrix between the two vectors corresponding to the subscripts, $\boldsymbol{b}$ represents the bias corresponding to its subscripts, and ϕ represents the activation function (usually the hyperbolic tangent function).

RNN has been successful due to its topology on the time dimension and is widely used in a variety of tasks like natural language processing. The RNN aims at processing sequence tagging of inputs by their internal state (memory) which enables the model to predict the current output conditioned on long-distance features. This makes the RNN applicable to tasks such as sequential handwriting recognition [33–35], Neural Machine Translation (NMT) [36–40], speech recognition [41,42] and object trajectory prediction [43]. However, RNN still suffers from a big problem, that is, the exponentially decreasing of the previous information due to the iterative formula of RNN. This is called gradient vanishing which makes RNN easily lose the ability to converge long-distance features [44,45]. The HELP should be able to extract maintain training information in the whole training process. Original RNN may make the HELP forgetful of the learning experience.

Therefore, LSTM [10] is proposed to avoid gradient vanishing. LSTM is the improved architecture of the RNN which is a kind of neural network that maintains a memory based on historical information. More than the standard architecture of RNN, LSTM introduces memory cells that control the internal state of LSTM in parallel (Fig. 1). When receiving the input sequence, the information will be accumulated in the cell, and the input gate activates at the same time. Forget gate controls whether to discard some previous information. Finally, the current output of the cell will be passed to the final state controlled by the output gate. The most

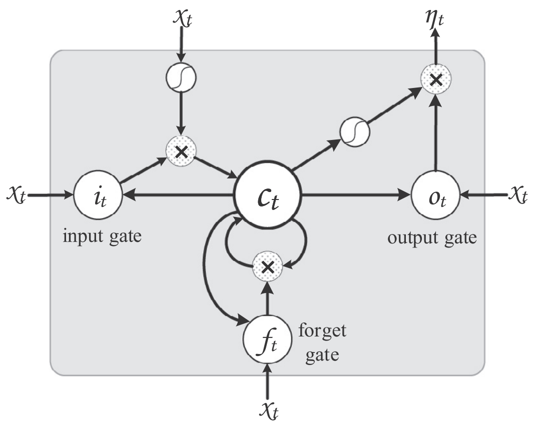


Fig. 1. Long Short-Term Memory Cell.

significant advantage of LSTM is its ability to control gates. Information will not disappear by the gradient iteration, because the important information will be trapped within the cell, and unimportant information can be forgotten by the forget gate which will not mingle with the useful part in the cell. As a result, LSTM has superiority in arranging long-distance dependencies from data.

Here the key equations of the gates in LSTM can be implemented as the following:

$$\begin{aligned} \boldsymbol{f}_t &= \sigma(\boldsymbol{x}_t \boldsymbol{W}_{xf} + \boldsymbol{\eta}_{t-1} \boldsymbol{W}_{\eta f} + \boldsymbol{c}_{t-1} \boldsymbol{W}_{cf} + \boldsymbol{b}_f) \\ \boldsymbol{i}_t &= \sigma(\boldsymbol{x}_t \boldsymbol{W}_{xi} + \boldsymbol{\eta}_{t-1} \boldsymbol{W}_{\eta i} + \boldsymbol{c}_{t-1} \boldsymbol{W}_{ci} + \boldsymbol{b}_i) \\ \tilde{\boldsymbol{c}}_t &= tanh(\boldsymbol{x}_t \boldsymbol{W}_{xc} + \boldsymbol{\eta}_{t-1} \boldsymbol{W}_{\eta c} + \boldsymbol{b}_c) \\ \boldsymbol{c}_t &= \boldsymbol{f}_t \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \tilde{\boldsymbol{c}}_t \\ \boldsymbol{o}_t &= \sigma(\boldsymbol{x}_t \boldsymbol{W}_{xo} + \boldsymbol{\eta}_{t-1} \boldsymbol{W}_{\eta o} + \boldsymbol{c}_t \boldsymbol{W}_{co} + \boldsymbol{b}_o) \\ \boldsymbol{\eta}_t &= \boldsymbol{o}_t tanh(\boldsymbol{c}_t) \end{aligned}$$

where $\boldsymbol{i}, \boldsymbol{f}, \boldsymbol{o}, \boldsymbol{c}$ denote input gate, forget gate, output gate and cell state with their subscripts presenting the sequence of time, respectively. σ presents the logistic sigmoid function ($\sigma(x) = \frac{1}{1+e^{-x}}$). All these gates are the same size as the hidden vector $\boldsymbol{\eta}$. The weight matrix $\boldsymbol{W}$ is the weight parameters with subscripts presenting the corresponding gate matrix, so as the bias vector $\boldsymbol{b}$ presenting the bias corresponding to its subscripts.

LSTM has started to revolutionize different researching areas including NMT, speech recognition [46,47] and other sequence tagging prediction tasks [48–50]. In our work, the HELP is proposed to make full use of the LSTM's excellent ability to extract and utilize past information. We put every deep neural network thread's hyperparameter sequence and its performance sequence into the RNN during the parallel training. We aim to get a predictor that can output the performance corresponding to the given hyperparameter sequence. The predictor will guide each thread during the parallel training, replacing the original random exploration process with a new one. For every deep neural network at every training iteration, a certain number of points in the hyperparameter space will be generated and the one yielding the best performance will be chosen by the predictor. In this way, the hyperparameter exploration process is more likely to find the point with better performance and becomes more efficient.

4. Hyperparameter exploration LSTM-predictor

The training of a deep neural network can be formulated as an optimization problem of finding a set of connection weights $\boldsymbol{w}$ for a defined neural network model with a set of hyperparameters $\boldsymbol{h}$

which minimize a pre-defined loss function. With the optimization process $\mathcal{O}$, each optimization iteration can be defined as follows:

$$\boldsymbol{w}_n = \mathcal{O}(\boldsymbol{w}_{n-1}, \boldsymbol{h}_n) \tag{1}$$

When each optimizing iteration in the whole training process uses a set of hyperparameters $\boldsymbol{h}_n$ at the n^{th} optimizing step, the final weights after n training steps ($\boldsymbol{w}_n$) can be expressed as follows:

$$\begin{aligned} \boldsymbol{w}_n &= \mathcal{O}(\boldsymbol{w}_{n-1}, \boldsymbol{h}_n) \\ &= \mathcal{O}(\mathcal{O}(\boldsymbol{w}_{n-2}, \boldsymbol{h}_{n-1}), \boldsymbol{h}_n) \\ &= \mathcal{O}(\ldots \mathcal{O}(\mathcal{O}(\boldsymbol{w}_0, \boldsymbol{h}_1), \boldsymbol{h}_2), \ldots, \boldsymbol{h}_n) \end{aligned} \tag{2}$$

Therefore, the optimization of hyperparameter is to find the best $\boldsymbol{h}$ in every step of the training process. However, the ideal goal cannot be realized directly by neural network optimization. But we hope to get closer to our goal by learning a similar, workable goal. That is, learning from a higher level, i.e. learning to learn to approach the goal that cannot be reached directly [3]. This is the concept that the HELP learns from the past learning experiences to adjust hyperparameters of the next iteration. At the optimization step n, there are T deep neural network threads. Without losing generality, we use one of the threads to explain the procedures of deciding the next exploratory hyperparameter $\boldsymbol{h}$ and all the T threads execute the same procedures. For a thread, several candidate hyperparameters are generated based on the current hyperparameter $\boldsymbol{h}_{n-1}$ at the last iteration. The key research problem is to select the candidate hyperparameter which is expected to yield the best $\boldsymbol{w}_n$ at the optimization step n. The original exploration process generates one candidate hyperparameter in each optimization iteration. For example, the new candidate hyperparameter in PBT [6] is the original hyperparameter after perturbation. In genetic algorithms, it is the new one generated by crossover and mutation of the old one. In PSO [23] algorithm it is the new particle location obtained by updating the speed and location of the old particle. The main disadvantage of this exploration process is that it cannot guarantee the new hyperparameter is improved compared to the previous one, much resources are wasted on such unsuccessful exploration.

Our proposed HELP is used to predict the fitness values v defined in Section 4.1 of candidate hyperparameters such that the exploration path for optimal $\boldsymbol{w}_n$ can be narrowed down to reduce both the time and the computational resource required. Previous hyperparameters are combined with the candidate hyperparameter to form a fixed-length (α) input vector $\boldsymbol{H}^* = (\boldsymbol{h}_{n-\alpha+1}, \boldsymbol{h}_{n-\alpha}, \ldots, \boldsymbol{h}_{n-1}, \boldsymbol{h}_n^*)$ for the HELP. The HELP outputs a predicted $\tilde{v}_n$ as the evaluation of the candidate hyperparameter $\boldsymbol{h}_n^*$ (Note that $\boldsymbol{h}_n^*$ is not the real hyperparameter being used to train the deep neural network, it is a candidate hyperparameter generated by original evolution strategy method that the HELP applies to). Then, the candidate hyperparameter $\boldsymbol{h}_n^*$ yielding the largest $\tilde{v}_n$ will be selected as the real training hyperparameter $\boldsymbol{h}_n$.

After all T exploration threads completes finding their new hyperparameters ($\boldsymbol{h}_n$), each of the T neural networks (corresponding to a thread) updates their new $\boldsymbol{w}_n$ using $O(\boldsymbol{w}_{n-1}, \boldsymbol{h}_n)$. An optimization iteration is completed.

The HELP will be put into use after the first $(\alpha + 1)$ optimization steps when enough data is collected to train the HELP. In the first α optimization steps, the original exploration process of the evolutionary strategy method is carried out rather than using the HELP.

To fully utilize the newly obtained exploration information from the last optimization step, the HELP updates its knowledge incrementally at the beginning of each optimization step. The incremental update of the HELP will be introduced in Section 4.2.

4.1. Evaluation of an optimization step

A neural network trained using $\boldsymbol{h}_n$ at the n^{th} optimization step has weight parameters $\boldsymbol{w}_n$ and its performance estimate is expressed as $\mathcal{E}(\boldsymbol{w}_n)$. Then, the fitness of the n^{th} optimization step is given as follows:

$$v_n = \begin{cases} \frac{\mathcal{E}(\boldsymbol{w}_n) - \mathcal{E}(\boldsymbol{w}_{n-1})}{maxeval - \mathcal{E}(\boldsymbol{w}_{n-1})}, \mathcal{E}(\boldsymbol{w}_n) - \mathcal{E}(\boldsymbol{w}_{n-1}) > 0 \\ \mathcal{E}(\boldsymbol{w}_n) - \mathcal{E}(\boldsymbol{w}_{n-1}), \mathcal{E}(\boldsymbol{w}_n) - \mathcal{E}(\boldsymbol{w}_{n-1}) \le 0 \end{cases} \tag{3}$$

where $maxeval$ denotes the maximum possible value of $\mathcal{E}(\boldsymbol{w}_n)$ and is a known constant prior to optimization, e.g. 1.0 for accuracy of image classification and the maximum Inception score [51] of a Generative Adversarial Net (GAN) [52].

Instead of finding the best $\boldsymbol{w}_n$, we aim to find the largest improvement in each step. Given the fact that improvement in later stages is usually smaller yet more difficult and important than that of earlier stages', Eq. (3) focuses on the improvement of the current step to maximize possible improvement. In experiments, the approximation of the evaluation value is obtained by testing the performance on a tiny validation set as in other guided hyperparameter tuning methods.

4.2. Incremental update of HELP

The architecture of the HELP is shown in Fig. 2. The HELP consists of two stacked LSTM cells and two fully-connected layers. The number of hidden units 32 is determined by the number of hyperparameters to be tuned. A detailed discussion is given in Section 5.4 The concatenate input hyperparameter sequence forms a matrix of shape $(\alpha, len(\boldsymbol{h}))$, the first fully-connected layer extracts high-level representations from different hyperparameters and outputs the data of shape $(\alpha, 32)$, which is regarded as a sequence of vectors in length α and each vector is of shape (32) by the LSTM network. The final output will be a sequence of numbers, the last number in the sequence is the predicted $\tilde{v}_n^*$. The fitness value v_n is defined in Section 4.1. The HELP gets hyperparameter sequence plus a candidate point $\boldsymbol{h}_n^*$ as input, and outputs the prediction of the fitness value of current $\boldsymbol{h}_n^*$. This step will replicate for each candidate hyperparameter. The HELP is a tiny neural network architecture for finding the maximum $\tilde{v}_n^*$ and does not require accurate prediction of their values.

Let δ be a natural number smaller than n. At the beginning of the optimization step n, the HELP takes $\boldsymbol{H}(\delta)$ as the input vector and learns to output a vector of $\tilde{\boldsymbol{V}}(\delta)$ = via a minimization of the difference between the predicted $\tilde{\boldsymbol{V}}(\delta)$ and the real $\boldsymbol{V}(\delta)$ using a

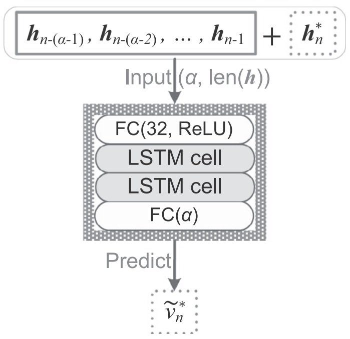

$\boldsymbol{h}_{n-(\alpha-1)}, \boldsymbol{h}_{n-(\alpha-2)}, \ldots, \boldsymbol{h}_{n-1}$ + $\boldsymbol{h}_n^*$
Input $(\alpha, \text{len}(\boldsymbol{h}))$
FC(32, ReLU)
LSTM cell
LSTM cell
FC(α)
Predict
$\tilde{v}_n^*$

Fig. 2. Predicting process flow chart. $\boldsymbol{\alpha}$ is the input length, $\text{len}(\boldsymbol{h})$ is the number of the hyperparameters to be tuned. The $\boldsymbol{h}_n^*$ with the highest predicted fitness value $\tilde{v}_n$ will be chosen.

loss function l defined in Eq. (5). The detailed data structures of $\mathbf{H}(\delta), \tilde{\mathbf{V}}(\delta)$ and $\mathbf{V}(\delta)$ are defined in Eq. (4).

$$\begin{cases} \boldsymbol{H}(\delta) = \left(\boldsymbol{h}_{(n-\delta)+1}, \boldsymbol{h}_{(n-\delta)+2}, \ldots, \boldsymbol{h}_{(n-\delta)+\alpha-1}, \boldsymbol{h}_{(n-\delta)+\alpha}\right) \\ \widetilde{\boldsymbol{V}}(\delta) = \left(\tilde{v}_{(n-\delta)+1}, \tilde{v}_{(n-\delta)+2}, \ldots, \tilde{v}_{(n-\delta)+\alpha-1}, \tilde{v}_{(n-\delta)+\alpha}\right) \\ \boldsymbol{V}(\delta) = \left(v_{(n-\delta)+1}, v_{(n-\delta)+2}, \ldots, v_{(n-\delta)+\alpha-1}, v_{(n-\delta)+\alpha}\right) \end{cases} \tag{4}$$

The training process of the HELP is shown in Fig. 3. The HELP is trained at each iteration before the deep neural network threads get trained. At optimization step n, there are $(n-1)$ pairs of selected hyperparameters and corresponding v values for each of the T deep neural network threads. For each thread, a consecutive sequence of hyperparameters with a length of α is selected from existing hyperparameters, i.e. $\boldsymbol{H}(\delta)$ where δ is randomly selected between $[\alpha, n]$, and its corresponding $\boldsymbol{V}(\delta)$ is selected to calculate training objective function value for the incremental supervised learning of the HELP. Therefore, T training samples are used at each training iteration, and the incremental learning step repeats for a pre-selected number of times.

The loss function of the HELP has three parts. For the predicted $\widetilde{\boldsymbol{V}}(\delta)$ and the real $\boldsymbol{V}(\delta)$, l_1 is designed to minimize the margin of error, l_2 for maximizing the covariance and l_3 is the regularization term. The loss function l is defined as follows:

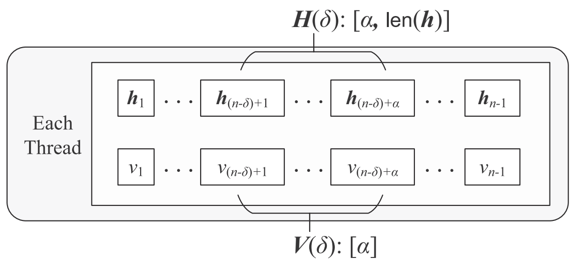
$$l = \lambda_1 l_1 + \lambda_2 l_2 + \lambda_3 l_3 \tag{5}$$

$$l_1 = E\left(\left(\widetilde{\boldsymbol{V}}(\delta) - \boldsymbol{V}(\delta)\right)^2\right) \tag{6}$$

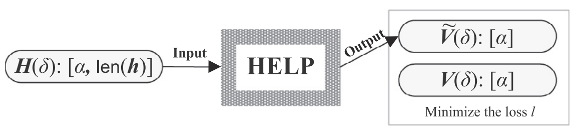
$$l_2 = \left(1 - r\left(\widetilde{\boldsymbol{V}}(\delta), \boldsymbol{V}(\delta)\right)\right)^2 \tag{7}$$

$$r\left(\widetilde{\boldsymbol{V}}(\delta), \boldsymbol{V}(\delta)\right) = \frac{\sum_{i=1}^{\alpha}\left(E\left(\widetilde{\boldsymbol{V}}(\delta)\right) - \tilde{v}_{(n-\delta)+i}\right)\left(E(\boldsymbol{V}(\delta)) - v_{(n-\delta)+i}\right)}{\sqrt{\sum_{i=1}^{\alpha}\left(E\left(\widetilde{\boldsymbol{V}}(\delta)\right) - \tilde{v}_{(n-\delta)+i}\right)^2}\sqrt{\sum_{i=1}^{\alpha}\left(E(\boldsymbol{V}(\delta)) - v_{(n-\delta)+i}\right)^2}} \tag{8}$$

$$l_3 = -\log_{10}\left(1 + \frac{sgn\left(\widetilde{\boldsymbol{V}}(\delta)\right)sgn(\boldsymbol{V}(\delta))^{\top}}{2\alpha}\right) \tag{9}$$



(a) Selecting sequences of hyperparameters and improving values.


$H(\delta)$: [α, len(h)]
Input
HELP
Output
$\widetilde{V}(\delta)$: [α]
$V(\delta)$: [α]
Minimize the loss l

(b) Minimizing the loss between the predictor output $\widetilde{V}(\delta)$ and real improving values sequence $V(\delta)$.

Fig. 3. Training process of the HELP. $\boldsymbol{H}(\delta)$ and $\boldsymbol{V}(\delta)$ are obtained from each deep neural network thread's history records. The HELP learns to correctly predict $\boldsymbol{V}(\delta)$ given $\boldsymbol{H}(\delta)$ for each thread..

where λ_1, λ_2, and λ_3 denote three trade-off parameters which are set to be 10000, 1, and 1, respectively, in the experiments. Here α denotes the length of $\widetilde{\boldsymbol{V}}(\delta)$ and $\boldsymbol{V}(\delta)$. The loss function l in Eq. (5) is optimized by the Adam Optimizer [53]. The training of the HELP is performed prior to the search of hyperparameters in all iterations. Then, the HELP guides the search of each deep neural network for this training iteration using the latest training information.

The whole process in the exploration step using the HELP is expressed in Algorithm 1 while the training process using the HELP is illustrated in Fig. 4. Candidate hyperparameters are generated by the original evolutionary strategy method, then the HELP chooses the next hyperparameter to explore. The HELP is a model-agnostic structure because the HELP can be added to most parallel hyperparameter tuning methods. The ablation studies are done in Section 5.4 to illustrate the rationality of such configuration.

Algorithm 1 Hyperparameter Exploration LSTM-Predictor

```
1: for each iteration of the training process do
2:     for each iteration of training the HELP do
3:         for each deep neural network thread do
4:             Determine a beginning point of the training
               sequence $\delta \in [\alpha+1, n]$ by Sigmoid-like distribution
5:             Get deep neural network's hyperparameter
               records $\boldsymbol{H}(\delta)$ and corresponding performance records $\boldsymbol{V}(\delta)$
6:         end for
7:         Optimizing $l\left(\boldsymbol{V}(\delta), \widetilde{\boldsymbol{V}}(\delta)\right)$ for all the deep neural
           network threads, where $\widetilde{\boldsymbol{V}}(\delta)$ is the predicting values that
           the HELP gives
8:     end for
9:     for each deep neural network thread do
10:        Generate multiple candidate hyperparameters $\boldsymbol{h}_n^*$
           by original evolutionary strategy method    ▹ Both
           mutation and crossover are random, so multiple runs yield
           different individuals.
11:        for each $\boldsymbol{h}_n^*$ do
12:            Previous hyperparameters are combined
               with the candidate hyperparameter to form a fixed input
               vector $\left(\boldsymbol{h}_{n-\alpha+1}, \boldsymbol{h}_{n-\alpha+2}, \ldots, \boldsymbol{h}_{n-1}, \boldsymbol{h}_n^*\right)$
13:            The HELP receives the input vector and
               outputs the prediction $\tilde{v}_n$ of the candidate hyperparameter
               $\boldsymbol{h}_n^*$
14:            Choose the $\boldsymbol{h}_n^*$ yielding the largest $\tilde{v}_n$ as the
               real $\boldsymbol{h}_n$
15:        end for
16:        Train the neural network with $\boldsymbol{h}_n$
17:    end for
18: end for
```

Only one HELP is used to guide the searching of hyperparameters. So the computational overhead yielded by HELP is a fixed amount and will not be affected by the size of the neural networks guided by HELP. Therefore, the general HELP structure is more suitable for larger deep neural networks.

Fig. 5 presents how the HELP guides the hyperparameter tuning and accelerates the whole training process. We show the exploration process on different 2-D test functions to illustrate how the HELP guides the exploration in different situations. We show that in the different exploration areas, the HELP can develop different exploration strategies and find good results. The red trace and the blue trace present the random exploration and exploration with the HELP in the evolutionary strategy, respectively. θ_1 and θ_2 presents two hyperparameters and the whole plane is the

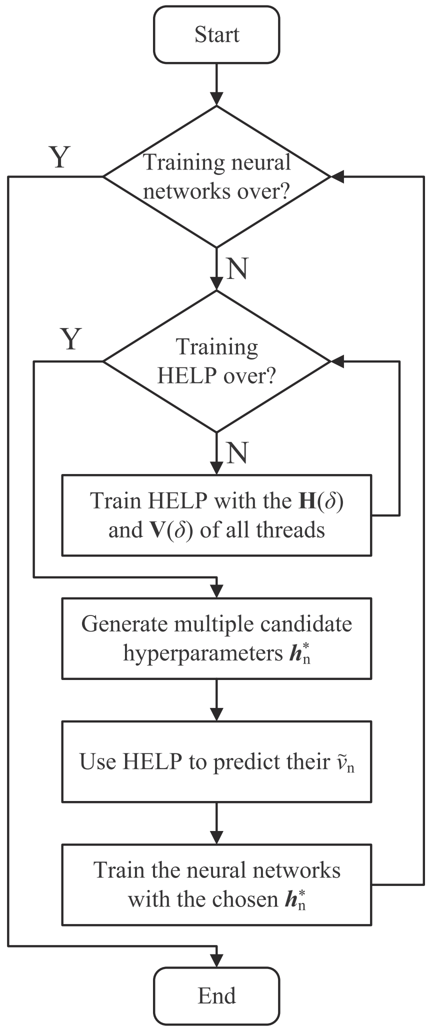

Start
Training neural networks over?
Y
N
Training HELP over?
Y
N
Train HELP with the $\mathbf{H}(\delta)$ and $\mathbf{V}(\delta)$ of all threads
Generate multiple candidate hyperparameters $\boldsymbol{h}_n^*$
Use HELP to predict their $\tilde{v}_n$
Train the neural networks with the chosen $\boldsymbol{h}_n^*$
End

Fig. 4. The block diagram of the HELP.

hyperparameter space. The trails with lighter color present earlier exploring space, and darker color trails present latter exploring space. As the exploration process goes on, the HELP can learn more information during the explored positions, so it guides the exploration direction by predicting the fitness value of the next hyperparameter to be chosen.

The corresponding test functions are given by the following:

1. Ackley function

$$f_1(x,y) = -20\exp\left(-0.2\sqrt{0.5(x^2+y^2)}\right) - \exp(0.5(\cos 2\pi x + \cos 2\pi y)) + e + 20 \tag{10}$$

2. Hölder table function

$$f_2(x,y) = -\left|\sin x \cos y \exp\left(\left|1 - \frac{\sqrt{x^2+y^2}}{\pi}\right|\right)\right| \tag{11}$$

3. Rastrigin function

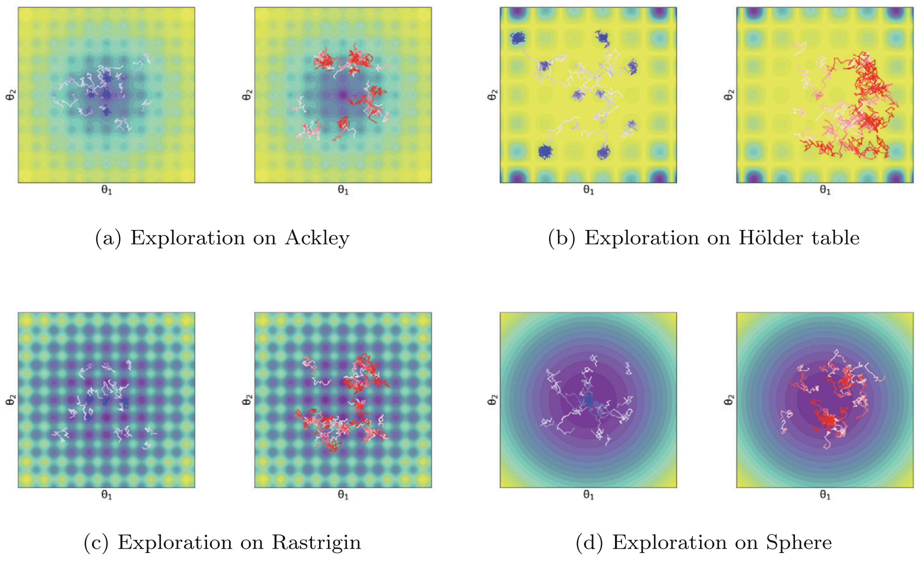
$$f_3(x,y) = x^2 - 10\cos 2\pi x + y^2 - 10\cos 2\pi y + 20 \tag{12}$$

4. Sphere function

$$f_4(x,y) = x^2 + y^2 \tag{13}$$

5. Experiments

In experiments, the HELP is applied to different neural network structures, including the classification task and the data generation task. We compare exploring new hyperparameters by the HELP with the original exploration process in the evolutionary strategy. We combined our HELP structure with PBT [6] evolutionary strategy method. In its *exploit* step, truncation selection is used. The later half threads are replaced by the first half threads both in hyperparameters and weight parameters. The specific experiments of hyperparameter tuning are described for Convolutional Neural Network (CNN) [54] in Section 5.1 with same accuracy rate goals in the testing set among traditional random search [5], PSO [24], PBT [6] and PBT with HELP. We also observe the changes in learn-



(a) Exploration on Ackley

(b) Exploration on Hölder table

(c) Exploration on Rastrigin

(d) Exploration on Sphere

Fig. 5. Optimization of toy examples using the HELP (blue) and random exploring (red). The deeper color area indicates smaller values in the contour map. The toy examples show the high efficiency of the hyperparameter tuning process with the HELP. The exploring routes of the HELP is more targeted, and its exploring areas are more focused on the well-perform areas, compare with original exploration wandering around the whole hyperparameter space. It reveals that the HELP learns to keep away from inefficient regions, which the other methods cannot do. Therefore, HELP finds better solutions than random exploring.

ing rate at each iteration while keeping other hyperparameters fixed to find the guiding characteristic of the HELP. In Section 5.2, we apply the HELP to the GAN [52] training whose training process is recognized as vulnerable and sensitive to hyperparameters. Then, we analyze the property and the difference in the training process with and without the HELP. We treat each deep neural network as a black box, only focus on its hyperparameter $\boldsymbol{h}$ as input and its performance v as output. The complexity inside the black box does not affect the tuning process. During the new hyperparameter exploration process, 100 candidate hyperparameters solutions will be generated, and HELP will pick the one yielding the best predicting result $\tilde{v}$. We replicate 10 times for each experiment and record the average results. We present the state-of-art results on a wide range of hyperparameter tuning problems by using the HELP. In Section 5.3, the computation resource used for the extra training of the HELP is shown. It shows that HELP is efficient given the improvements it makes. In Section 5.4, ablation experiments are shown to validate the design of the HELP's structure.

5.1. HELP in Convolutional Neural Network (CNN) hyperparameter tuning

CNN has first been introduced a long time ago [54]. Due to the rapid improvement of the computer hardware and the great success achieved by CNN in the computer vision field, larger and deeper neural network architectures based on CNN were introduced. This kind of deep neural network with shared weights architecture shows a great translation invariance trait in pattern recognition [8]. The structure of CNN also evolves from the early AlexNet [55] to recent variations of the Residual Network [56] with over 1000 layers.

In our experiment of the CNN hyperparameter tuning, we use a CNN architecture shown in Fig. 6. The default deep network configuration in the deep learning framework is used. As we focus on the hyperparameter tuning, the fixed configuration is only for fair comparison and makes the hyperparameters tuned by different methods to be the only variables affecting the results. Therefore, we fix the configuration and use the default configuration directly. Each layer follows a dropout operation [57] and a LeakyReLU activation function [58]. The last layer is the output layer.

In CNN training, we aim to find the most suitable hyperparameter to make the training process faster and more precise. Here the

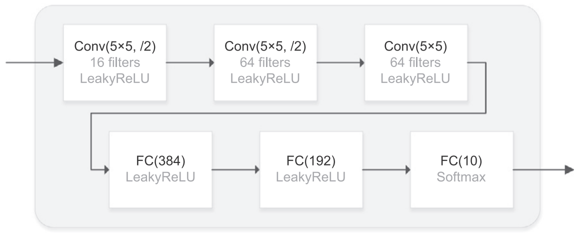


Fig. 6. Structure of CNN in the training process. $\mathrm{Conv}(5\times5,/2)$ presents a convolution layer with a 5×5 convolution kernel, appending a max-pooling layer with stride 2. $\mathrm{FC}(384)$ presents a fully-connected layer with 384 hidden units.

mnist [59] dataset is used to train the CNN by its training set and evaluate by its validation set. We split the *mnist* training set and randomly take 1000 samples as the baseline for evaluation as the validation set. The performance of a deep neural network is approximated by the performance of the validation set. Therefore, in the case that the sample distribution of the training set and the testing set is consistent, the random division can be adopted. The testing set is used to measure the real performance of the deep neural network. We set the goal accuracy rate at 95.00% on its real performance since the tuning phase of higher accuracy depends more on the luck of the initialization which cannot reflect the effect of the hyperparameter tuning models.

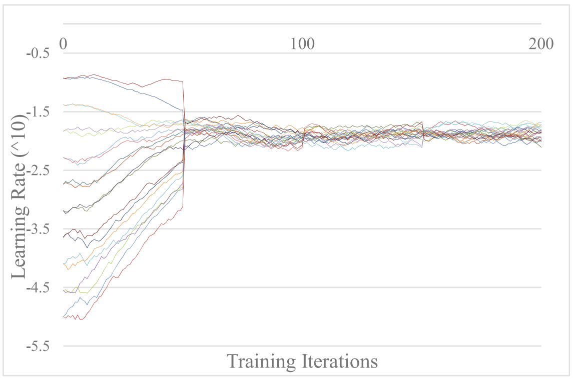
In CNN hyperparameter tuning, we consider hyperparameters including learning rate, dropout rate, and each layer's LeakyReLU alpha. In order to keep the same scale at each iteration, the batch size is fixed to 256. The learning rate of each initial thread is uniformly distributing on an exponential scale between $10^{-0.5}$ and 10^{-7}. The initial dropout rates are uniformly distributing between 0 (all drop) and 1 (no drop). Each layer's LeakyReLU alpha is uniformly distributing from 0 (simple ReLU) and 1 (without nonlinear activation function). The hyperparameters of each thread are randomly chosen from the hyperparameter range, 6 initial points are given where each point has 2 threads. In *explore* step of the evolutionary strategy, the hyperparameter perturbing rate is 0.04, the learning rate is on the scale of the exponential perturbation while the other hyperparameters are on the linear perturbation. In the optimization process, we use a mini-batch stochastic gradient descent optimizer. The *exploit* process will be taken every 50 training iterations. For PSO hyperparameter tuning method [24], we set the search parameter $\omega=\phi_1=\phi_2=0.5$ during the training. The fitness value of PSO is the current accuracy rate on the validation set.

Table 1 shows the average number of iterations that the training takes to get the corresponding accuracy rate. It shows that in the evolutionary strategy, hyperparameter exploration by the HELP can be more than twice as fast as the random exploring is at a relatively low accuracy rate. In the high accuracy rate required situation, exploring by the HELP can also provide faster speed than the random exploring method. For the PSO search, it is faster than the evolutionary strategy at the beginning, but its efficiency gets lower during the later stage compared to the evolutionary strategy. The main reason is that PSO keeps all the population without selection and reproduction, which makes the extreme individual threads occupy too much resource. The application of HELP in PSO has improved the efficiency of hyperparameter tuning. The guidance of the HELP makes PSO still have a large exploration range at the later stage. The later performance of PSO even exceeds the original evolutionary strategy. As for traditional random search, it is worse than both evolutionary strategy and PSO but not by much at low accuracy rate required situation. However, it soon shows a widening gap at the higher accuracy rate required situation, because it does not apply dynamic hyperparameters at each training iteration, nor rearrange more threads to the excellent-result positions in hyperparameter space.

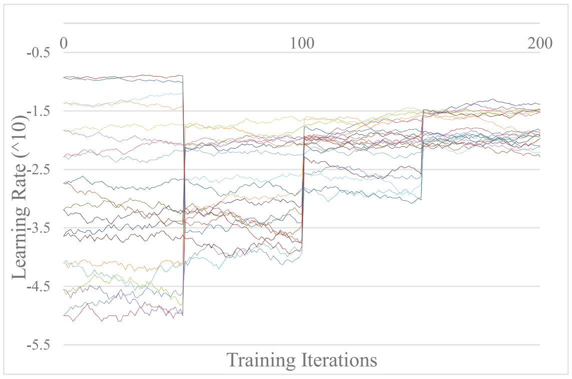
For hyperparameters in the neural network that can be divided into important hyperparameters and unimportant hyperparameters [5], the tuning of the latter one can hardly affect the training

Table 1
Iterations Required to Achieve the Accuracy.

Method\Acc	85.0%	86.0%	87.0%	88.0%	89.0%	90.0%	91.0%	92.0%	92.5%	93.0%	93.5%	94.0%	94.5%	95.0%
ES with HELP	**56.8**	**60.4**	**67.1**	**67.7**	**73.3**	**80.7**	**91.0**	**106.4**	**113.0**	**120.3**	**137.4**	**156.5**	**182.3**	**215.1**
ES	118.9	124.3	134.2	142.5	156.9	171.1	194.6	220.3	233.3	253.0	280.5	310.6	345.9	398.3
PSO with HELP	57.8	65.1	75.4	83.3	95.1	109.3	128.1	167.8	182.8	203.1	223.6	274.5	292.5	336.6
PSO	83.4	89.7	98.0	110.1	127.5	145.8	182.5	208.0	227.5	259.8	277.8	332.3	390.0	446.9
Random Search	137.9	149.8	165.8	183.5	211.1	249.6	303.1	365.0	399.3	451.9	521.3	609.6	726.9	875.8



(a) Learning rates by exploring with the HELP.



(b) Learning rates by random exploration.

Fig. 7. Learning rates of the evolutionary strategy with and without the HELP on CNN.

process. In other words, whether tuning the unimportant hyperparameters with or without the HELP structure has little effect on the experiment result. Thus, we focus on comparing tuning important hyperparameters. From previous experiments, we find that unimportant hyperparameters like dropout rate and LeakyReLU alpha have little impact on the training process unless they take extreme values.

In the CNN training with the accuracy rate goal, the learning rate mainly affects the improving speed of the accuracy rate [60]. Next, we trace the learning rates at each iteration which are shown as important hyperparameters in CNN training in the previous experiment. In this subsection, we initialize the learning rates of each individual between 10^{-1} and 10^{-5}, keeping the other hyperparameters fixed in order to avoid other factors disturbing (dropout rates 0.9 and each layers' LeakyReLU alpha 0.2). 10 initial points are given where each point has 2 threads. We aim to show and compare the learning rates (Fig. 7) to analyze the characteristics and the advantages of exploring with the HELP.

We can see that the learning rates are more likely to move to the direction that has the highest probability to gain improving value defined in Section 4.1. By adding the HELP structure to the random exploration process in the evolutionary strategy, the hyperparameter tuning process can converge faster due to the more efficient exploration.

5.2. HELP in Generative Adversarial Net (GAN) hyperparameter tuning

The GAN [52] is an unsupervised generative model trained via an adversarial process. Since it was first introduced, much work has been done to expand its applied range [61–63] and improve its performance [64–67]. GAN architecture has also been applied to wider fields such as style transfer by cycle GAN [68]. However, GAN architecture always suffers from an unstable training process due to the discrepant training of its two parts.

During the training process of GAN [52], the discriminator gets input samples from both the generator and the real data, being trained to predict whether the input is fake or real. The generator gets random noise input to map the noise distribution to the real data distribution. The training process reaches a saturation point by the time the discriminator cannot correctly identify the input source anymore.

Different from the CNN training, GAN is more unstable, hyperparameters especially learning rates will affect the training results for the GAN training process. Any larger impact on the two struggling parts may cause a collapse of the whole GAN. In traditional hyperparameter tuning, hyperparameters of the two parts of GAN are usually the same so that it is easy to control in the manual hyperparameters setting process. But this manual-added rule may not find the optimal solution of the hyperparameter pairs. In our hyperparameter tuning, we adjust hyperparameters of two parts independently, pushing them to find the optimal pairs by themselves.

In the GAN training, hyperparameter is tuned using the typical GAN architecture [64] and evaluated by the Inception score [51]. In our GAN experiment, each training iteration includes one optimization process for both the generator part and the discriminator part. We use alternative training at each optimization process, it means the network can be trained repeatedly a maximum of 5 times if the current part loss cannot meet the required relationship with the other loss. We compare the training process by achieving the goal Inception score. During the GAN training, we use *CIFAR-10* [69] dataset for comparison [66,6], with initial threads' learning rates distributing between $10^{-2.5}$ and 10^{-6}, given 8 initial points. The initial dropout rates and LeakyReLU alpha are also uniformly distributing between 0 and 1. We keep the batch size is fixed to 32 at each iteration.

The training process adopts Adam [53] optimizer. The *exploit* process will be taken every 40000 training iterations. The experiment result is shown in Table 2. Exploring with the HELP requires fewer training iterations than the original PBT [6] at each Inception score goal, the difference increases during the training process. With training iterations increasing, the HELP is stronger enough to give a credible prediction for the whole training threads to speed up the training process. On the other hand, original PSO and traditional random search show more inefficiency than they do in the CNN hyperparameter tuning experiment. This is because the hyperparameters in GAN do not only influence the training speed.

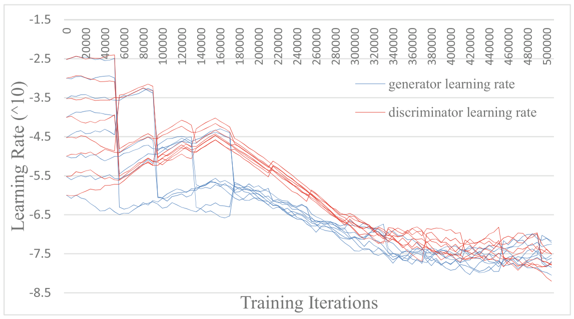
Table 2
Iterations Required to Achieve the Inception Score.

Method\Acc	5.0	5.5	6.0	6.5	7.0
ES with HELP	**142.8 k**	**150.0 k**	**166.8 k**	**172.2 k**	**198.6 k**
ES	152.4 k	153.6 k	178.8 k	217.2 k	229.2 k
PSO with HELP	158.8 k	196.8 k	207.6 k	282.0 k	341.6 k
PSO	190.8 k	282.0 k	353.1 k	445.2 k	508.8 k
Random Search	286.2 k	318.6 k	466.2 k	486.0 k	588.6 k

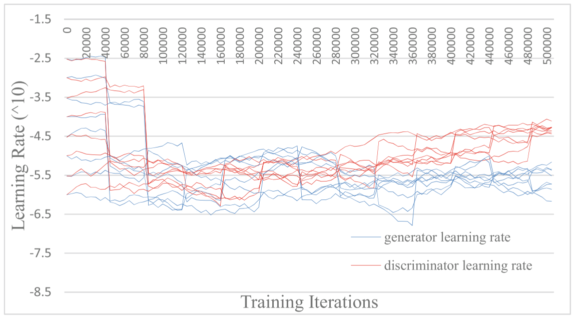
Since the GAN is more vulnerable, improper hyperparameters in GAN training may cause a collapse to the whole deep neural network. Traditional random searching without a selection process cannot drop the collapsed neural network and refresh the whole training threads, which makes it suffer the loss. Original PSO hyperparameter search applied in GAN also shows a poor result in the GAN hyperparameter tuning. It is even closer to the performance of the traditional random search in the later training stage. The main reason is that PSO converges prematurely during the GAN training process. In the PSO experiment, we observe that the hyperparameters of all threads converge rapidly. For the GAN training process, the optimal hyperparameters are dynamic at different stages. Original PSO is unable to explore the new hyperparameters adjustment strategy that should be adopted after its convergence. However, this changed after the use of the HELP. The performance of the PSO with HELP is close to the original evolutionary strategy under high Inception score requirements. PSO does not converge immediately, but it actively explores other positions in the hyperparameter space, thus avoiding the inefficient performance of the original PSO in the later stage. Due to the *exploit* process in the evolutionary strategy, it shows a better result than original PSO and traditional random search. With the HELP improving the random exploration process, a more efficient hyperparameter tuning speed can be achieved. The improvement on the Inception score is very slow in the late stage, so it is difficult to reflect the efficiency of the HELP. However, with the promotion of the target, ES with HELP gradually pulled away from the original ES. The HELP makes evolutionary strategies quickly find an appropriate hyperparameter adjustment strategy. Next, we show what strategy the HELP guides the deep neural networks to apply during the hyperparameter tuning of GAN.

As the typical schedule of GAN training, the learning rates of the discriminator and generator should be in a linearly exponential decay to gradually make the training more accurate. Fig. 8 presents the progress of the learning rates exploring with the HELP and the original evolutionary strategy's exploration process. It shows that exploring with the guidance of the HELP can quickly adopt this efficient learning rate schedule. Because the decay of the learning rate shows a larger fitness value than the other schedules, the HELP will allocate more probability to explore lower learning rates. On the other hand, the evolutionary strategy cannot directly improve such a training dilemma with completely random exploration. Its improvement only depends on the *exploit* step to drop the current poor-performance threads and copy from well-behaved ones.

Another discovery is that the optimal learning rates of the discriminator and the generator are not exactly the same. The discriminator learning rate should be slightly larger than the generator for some time, which is hard to find by manual hyperparameter optimization. Moreover, the HELP finds and applies this strategy one step ahead which speeds up the training process as a result.



(a) Learning rates by exploring with the HELP.



(b) Learning rates by random exploration.

Fig. 8. Learning rates of the evolutionary strategy with and without the HELP on GAN.

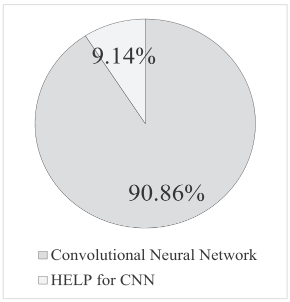
5.3. Computational resource for HELP

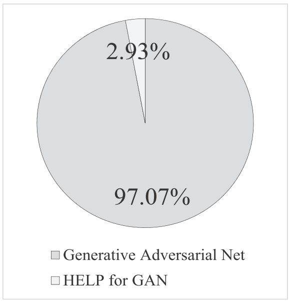
With the use of the HELP, the training process of the deep neural network can be divided into two phases: training deep neural networks and applying HELP to them. In this section, we record GPU loads for training deep neural networks in Sections 5.1 and 5.2, and how much additional computational resource is used by the HELP (Fig. 9). In this part, we record the computational resource-consuming ratios for both the first phase and the second phase. It yields the extra training time taken by the HELP.

Fig. 9 shows portions of computational resources used by training deep neural networks and applying the HELP for CNN and GAN, respectively. It shows that the HELP uses very few computational resources because it is a small structure comparing to deep neural networks and does not scale up when the deep neural network grows. So, the portion of computational resources used by the HELP will be smaller when the deep neural network is larger. Furthermore, the HELP selects the candidate hyperparameter yielding the largest predicted $\tilde{v}$, so an accurate prediction of v is not necessary. Therefore, the HELP does not require fine-tuning for highly accurate prediction and only needs to predict the direction of exploration for hyperparameter selection for deep neural networks.

5.4. Ablation studies

In this section, ablation studies are provided to validate the design of combining three loss functions and the selection of α. We train the same CNN in Section 5.1 with a random distributed





(a) The CNN resource ratio. (b) The GAN resource ratio.

Fig. 9. The computational resource HELP occupies in different kinds of deep neural network learning. This is the largest ratio that the HELP can occupy, since the training process of the HELP, which takes a great majority in HELP's resource consumption, can be cut down by parallelization.

Table 3
HELP's predict effect with different loss function.

Loss function	Spearman coefficient
l_1	0.5146 ± 0.0013
l_2	0.5161 ± 0.0005
l_3	0.0217 ± 0.0086
$l_1 + l_2$	0.5246 ± 0.0027
$l_1 + l_3$	0.5174 ± 0.0007
$l_2 + l_3$	0.5162 ± 0.0054
$l_1 + l_2 + l_3$	**0.5265 ± 0.0003**

hyperparameters at each training iteration, and record the hyperparameter sequence as $\boldsymbol{H} = (\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_n)$. Then its fitness value defined at Section 4.1 is recorded, all the fitness values in n iterations are concatenated as a sequence $\boldsymbol{V} = (v_1, v_2, \ldots, v_n)$. We generate 1000 records as training set (each record denotes as a pair $\{\boldsymbol{H}_{train}, \boldsymbol{V}_{train}\}$), and another 100 records as testing set (each record denotes as a pair $\{\boldsymbol{H}_{test}, \boldsymbol{V}_{test}\}$) for the HELP, each record has the length n of 150. These training and testing sets are only used in this section for ablation studies and not participated in any training of the HELP. The Spearman's rank correlation coefficient computed between $\boldsymbol{V}_{test}$ and the prediction of the HELP for $\boldsymbol{V}_{test}$ is adopted to judge the performance of them which are shown in Table 3.

Table 3 shows that the combination of all the loss parts yields the best results among individual parts. Moreover, the combination of all parts shows a smaller standard deviation which means a more stable training process of the HELP. This validates the proposed combination of the triple loss functions.

Besides loss function design, the choice of input sequence length for the HELP (α) also significantly influences the performance of the HELP. A longer input sequence means that the HELP needs longer deep neural network pre-train iterations to collect enough preprocessing data. A shorter one uses less pre-train iterations, but the prediction of the HELP may be inaccurate. To validate the choice of $\alpha = 10$ to be the most suitable choice, average predicting errors of different α values are tested using the aforementioned training and testing sets. Experimental results are shown in Fig. 10.

Testing results show the relation between input length and average error. With input length increasing, the average predicting error becomes smaller. Fig. 10 shows that when α is larger than 9, the average error stops sharply decreasing. So to take a balanced view to consider the trade-off between the number of pre-train iterations and predicting accuracy, we choose $\alpha = 10$ as the HELP's predicting length.

The first fully-connected layer of the HELP extracts high-level representations from different hyperparameters. Thus, the number of fully-connected layer units is related to the number of hyperparameters. Fig. 11 shows that average predicting errors do not

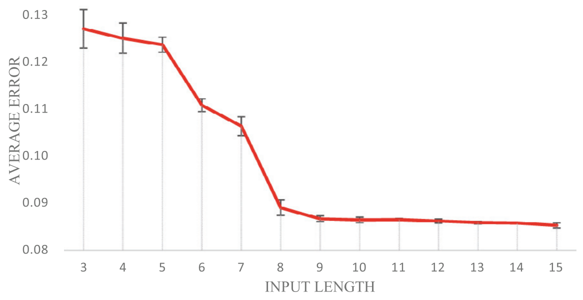


Fig. 10. Comparison of average predicting error with different input lengths (α) of the HELP. It shows that when the length of the input sequence is greater than 9, the HELP reaches a low prediction error.

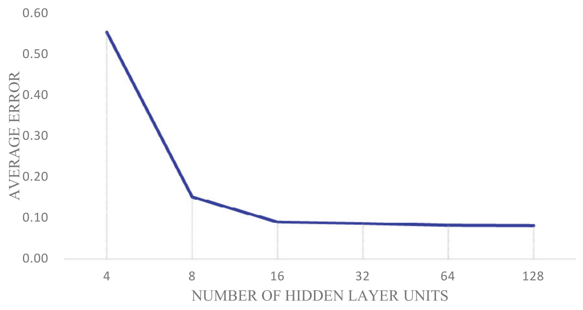


Fig. 11. Comparison of average predicting error with different numbers of hidden units of the HELP. The HELP reaches a low prediction error when the number of hidden units is greater than 16.

reduce significantly when the number of units is greater than 16. Under the consideration that more units may yield better representation but also cost more computations, 32 units are used in our experiments.

6. Conclusions

In this paper, we have proposed an LSTM-based structure to guide the hyperparameter tuning by changing the probabilities of the choices among different exploring directions in hyperparameter space so that it speeds up the training process. We have shown significant improvements in the hyperparameter tuning of different kinds of deep neural networks. The HELP structure predicts the performance of each hyperparameter point by using the information during the training process. Therefore, the exploration probability in each direction for each point on the hyperparameter plane is reallocated in the new training iteration of each deep neural network. In this way, the exploration process becomes more efficient and the deep neural network has a higher probability to find better hyperparameters.

The experimental results show that the hyperparameter tuning in neural networks would be faster and more stable if we replace the equiprobable random exploring process with the unequal probability-based exploring process judged from the previous training experience. By considering all the previous experience rather than just keeping well-behaved individuals, the HELP can learn from failure lessons and avoid making similar mistakes.

In our experiments, the HELP also shows a reasonable way of dealing with different kinds of deep networks: 1) in the CNN hyperparameter tuning process it tends to make the optimization step size converge to a fixed value during the exploration, 2) in GAN hyperparameter tuning process it tries to reduce the optimization step size to get a more meticulous treatment. This means that the HELP can excavate specific hyperparameter tuning strategies for different deep neural networks based on its training information.

However, the HELP may suffer from collapse due to the extreme hyperparameter initialization. How to deal with this vicious circle reasonably and get a more stable HELP structure is our future research goal. Moreover, the selection of hyperparameters for neural architecture design is also an important part of hyperparameter optimization. How to make the HELP fit in this field will be the subject of future investigations.

CRediT authorship contribution statement

Wendi Li: Methodology, Software, Writing - original draft. **Wing W. Y. Ng:** Conceptualization, Resources, Supervision, Funding acquisition, Writing - review & editing. **Ting Wang:** Validation, Formal analysis, Writing - original draft. **Marcello Pelillo:** Supervision, Writing - review & editing. **Sam Kwong:** Supervision, Writing - review & editing.

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.

Acknowledgment

This work was supported by the National Natural Science Foundation of China under Grants 61876066 and 61672443, Guangdong Province Science and Technology Plan Project (Collaborative Innovation and Platform Environment Construction) 2019A050510006, the National Training Program of Innovation and Entrepreneurship for College Students 201910561071, and Hong Kong RGC General Research Funds under Grant 9042038 (CityU 11205314) and Grant 9042322 (CityU 11200116).

References

[1] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, et al., Evolving deep neural networks, in: Artificial Intelligence in the Age of Neural Networks and Brain Computing, Elsevier, 2019, pp. 293–312.
[2] Z.-Q. Zhao, P. Zheng, S.-T. Xu, X. Wu, Object detection with deep learning: A review, IEEE Transactions on Neural Networks and Learning Systems 30 (11) (2019) 3212–3232.
[3] M. Feurer, F. Hutter, Hyperparameter optimization, Automated Machine Learning, Springer (2019) 3–33.
[4] H.C. Law, P. Zhao, L.S. Chan, J. Huang, D. Sejdinovic, Hyperparameter learning via distributional transfer, Advances in Neural Information Processing Systems (2019) 6801–6812.
[5] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, Journal of Machine Learning Research 13 (Feb) (2012) 281–305.
[6] M. Jaderberg, V. Dalibard, S. Osindero, W.M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, et al., Population based training of neural networks, arXiv preprint arXiv:1711.09846.
[7] I. Loshchilov, F. Hutter, Cma-es for hyperparameter optimization of deep neural networks, arXiv preprint arXiv:1604.07269.
[8] S.R. Young, D.C. Rose, T.P. Karnowski, S.-H. Lim, R.M. Patton, Optimizing deep learning hyper-parameters through an evolutionary algorithm, in: Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments, 2015, pp. 1–5.
[9] B. Doerr, C. Doerr, T. Kötzing, Static and self-adjusting mutation strengths for multi-valued decision variables, Algorithmica 80 (5) (2018) 1732–1768.
[10] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Computation 9 (8) (1997) 1735–1780.
[11] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, Y. Bengio, An empirical evaluation of deep architectures on problems with many factors of variation, in: Proceedings of the 24th International Conference on Machine Learning, 2007, pp. 473–480.
[12] G.E. Hinton, A practical guide to training restricted boltzmann machines, in: Neural Networks: Tricks of the Trade, Springer, 2012, pp. 599–619.
[13] R.K. Srivastava, K. Greff, J. Schmidhuber, Training very deep networks, Advances in Neural Information Processing Systems (2015) 2377–2385.
[14] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[15] J. Snoek, H. Larochelle, R.P. Adams, Practical bayesian optimization of machine learning algorithms, Advances in Neural Information Processing Systems (2012) 2951–2959.
[16] J.S. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyper-parameter optimization, Advances in Neural Information Processing Systems (2011) 2546–2554.
[17] F. Hutter, H.H. Hoos, K. Leyton Brown, Sequential model-based optimization for general algorithm configuration, in: International Conference on Learning and Intelligent Optimization, 2011, pp. 507–523.
[18] X. Zhang, X. Chen, L. Yao, C. Ge, M. Dong, Deep neural network hyperparameter optimization with orthogonal array tuning, International Conference on Neural Information Processing, Springer (2019) 287–295.
[19] A. Shah, Z. Ghahramani, Parallel predictive entropy search for batch global optimization of expensive objective functions, Advances in Neural Information Processing Systems (2015) 3330–3338.
[20] E.C. Garrido-Merchán, D. Hernández-Lobato, Dealing with categorical and integer-valued variables in bayesian optimization with gaussian processes, Neurocomputing 380 (2020) 20–35.
[21] T.T. Joy, S. Rana, S. Gupta, S. Venkatesh, Batch bayesian optimization using multi-scale search, Knowledge-Based Systems 187 (2020) 104818.
[22] M.T. Young, J.D. Hinkle, R. Kannan, A. Ramanathan, Distributed bayesian optimization of deep reinforcement learning algorithms, Journal of Parallel and Distributed Computing 139 (2020) 43–52.
[23] J. Kennedy, Particle swarm optimization, in: Encyclopedia of Machine Learning, Springer, 2011, pp. 760–766.
[24] P.R. Lorenzo, J. Nalepa, L.S. Ramos, J.R. Pastor, Hyper-parameter selection in deep neural networks using parallel particle swarm optimization, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, 2017, pp. 1864–1871.
[25] F. Ye, Particle swarm optimization-based automatic parameter selection for deep neural networks and its applications in large-scale and high-dimensional data, PloS one 12 (12) (2017) e0188746.
[26] S. Lessmann, R. Stahlbock, S.F. Crone, Optimizing hyperparameters of support vector machines by genetic algorithms, in: IC-AI, 2005, pp. 74–82.
[27] Y.-L. Li, Z.-H. Zhan, Y.-J. Gong, W.-N. Chen, J. Zhang, Y. Li, Differential evolution with an evolution path: A deep evolutionary algorithm, IEEE Transactions on Cybernetics 45 (9) (2015) 1798–1810.
[28] O.E. David, I. Greental, Genetic algorithms for evolving deep neural networks, Proceedings of the Genetic and Evolutionary Computation Conference (2014) 1451–1452.
[29] Y.-J. Gong, J. Zhang, Y. Zhou, Learning multimodal parameters: A bare-bones niching differential evolution approach, IEEE Transactions on Neural Networks and Learning Systems 29 (7) (2017) 2944–2959.
[30] C.-H. Chen, C.-B. Liu, Reinforcement learning-based differential evolution with cooperative coevolution for a compensatory neuro-fuzzy controller, IEEE Transactions on Neural Networks and Learning Systems 29 (10) (2017) 4719–4729.
[31] J. Lehman, J. Chen, J. Clune, K.O. Stanley, Safe mutations for deep and recurrent neural networks through output gradients, Proceedings of the Genetic and Evolutionary Computation Conference (2018) 117–124.
[32] T. Salimans, J. Ho, X. Chen, S. Sidor, I. Sutskever, Evolution strategies as a scalable alternative to reinforcement learning, arXiv preprint arXiv:1703.03864.
[33] D. Tao, X. Lin, L. Jin, X. Li, Principal component 2-d long short-term memory for font recognition on single chinese characters, IEEE Transactions on Cybernetics 46 (3) (2016) 756–765.
[34] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, J. Schmidhuber, A novel connectionist system for unconstrained handwriting recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence 31 (5) (2009) 855–868.
[35] V. Carbune, P. Gonnet, T. Deselaers, H.A. Rowley, A. Daryin, M. Calvo, L.-L. Wang, D. Keysers, S. Feuz, P. Gervais, Fast multi-language lstm-based online handwriting recognition, International Journal on Document Analysis and Recognition (IJDAR) (2020) 1–14.
[36] L. Bai, W. Liu, A practice on neural machine translation from indonesian to chinese, in: Recent Trends in Intelligent Computing, Communication and Devices, Springer, 2020, pp. 33–38.
[37] B. Zhang, D. Xiong, J. Su, H. Duan, A context-aware recurrent encoder for neural machine translation, IEEE/ACM Transactions on Audio, Speech and Language Processing 25 (12) (2017) 2424–2432.
[38] J. Xie, R. Yan, S. Xiao, L. Peng, M.T. Johnson, W.-Q. Zhang, Dynamic temporal residual learning for speech recognition, in: ICASSP 2020–2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2020, pp. 7709–7713.
[39] P.C. Vashist, A. Pandey, A. Tripathi, A comparative study of handwriting recognition techniques, in: 2020 International Conference on Computation, Automation and Knowledge Management (ICCAKM), IEEE, 2020, pp. 456–461.
[40] B. Zhang, D. Xiong, J. Xie, J. Su, Neural machine translation with gru-gated attention model, IEEE Transactions on Neural Networks and Learning Systems.
[41] H. Sak, A. Senior, F. Beaufays, Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition, arXiv preprint arXiv:1402.1128.
[42] X. Li, X. Wu, Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition, International Conference on Acoustics, Speech and Signal Processing (2015) 4520–4524.
[43] L. Wang, L. Zhang, Z. Yi, Trajectory predictor by using recurrent neural networks in visual tracking, IEEE Transactions on Cybernetics 47 (10) (2017) 3172–3183.
[44] R. Pascanu, T. Mikolov, Y. Bengio, On the difficulty of training recurrent neural networks, in: International Conference on Machine Learning, 2013, pp. 1310–1318.
[45] Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE Transactions on Neural Networks 5 (2) (1994) 157–166.
[46] S. Fernández, A. Graves, J. Schmidhuber, An application of recurrent neural networks to discriminative keyword spotting, in: International Conference on Artificial Neural Networks, 2007, pp. 220–229.
[47] A. Graves, A. Mohamed, G. Hinton, Speech recognition with deep recurrent neural networks, International Conference on Acoustics, Speech and Signal Processing (2013) 6645–6649.
[48] T. Ergen, S.S. Kozat, Online training of lstm networks in distributed systems for variable length data sequences, IEEE Transactions on Neural Networks and Learning Systems 29 (10) (2017) 5159–5165.

[49] T. Ergen, S.S. Kozat, Efficient online learning algorithms based on lstm neural networks, IEEE Transactions on Neural Networks and Learning Systems 29 (8) (2017) 3772–3783.
[50] G. Zhu, L. Zhang, L. Yang, L. Mei, S.A.A. Shah, M. Bennamoun, P. Shen, Redundancy and attention in convolutional lstm for gesture recognition, IEEE Transactions on Neural Networks and Learning Systems.
[51] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, Improved techniques for training gans, Advances in Neural Information Processing Systems (2016) 2234–2242.
[52] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, Advances in Neural Information Processing Systems (2014) 2672–2680.
[53] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: International Conference on Learning Representations, 2014, pp. 1–13.
[54] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.
[55] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Advances in Neural Information Processing Systems (2012) 1097–1105.
[56] G. Huang, Y. Sun, Z. Liu, D. Sedra, K.Q. Weinberger, Deep networks with stochastic depth, European Conference on Computer Vision (2016) 646–661.
[57] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, The Journal of Machine Learning Research 15 (1) (2014) 1929–1958.
[58] A.L. Maas, A.Y. Hannun, A.Y. Ng, Rectifier nonlinearities improve neural network acoustic models, Proceedings of the International Conference on Machine Learning 30 (1) (2013) 3.
[59] Y. LeCun, C. Cortes, C. Burges, Mnist handwritten digit database, AT&T Labs [Online]. Available: http://yann.lecun.com/exdb/mnist 2.
[60] A.G. Baydin, R. Cornish, D.M. Rubio, M. Schmidt, F. Wood, Online learning rate adaptation with hypergradient descent, arXiv preprint arXiv:1703.04782.
[61] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A.P. Aitken, A. Tejani, J. Totz, Z. Wang, et al., Photo-realistic single image super-resolution using a generative adversarial network, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 2 (3) (2017) 4.
[62] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, P. Abbeel, Interpretable representation learning by information maximizing generative adversarial nets, Infogan: Advances in Neural Information Processing Systems (2016) 2172–2180.
[63] F. Liu, L. Jiao, X. Tang, Task-oriented gan for polsar image classification and clustering, IEEE Transactions on Neural Networks and Learning Systems.
[64] M. Arjovsky, S. Chintala, L. Bottou, Wasserstein generative adversarial networks, in: International Conference on Machine Learning, 2017, pp. 214–223.
[65] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, A.C. Courville, Improved training of wasserstein gans, Advances in Neural Information Processing Systems (2017) 5767–5777.
[66] A. Radford, L. Metz, S. Chintala, Unsupervised representation learning with deep convolutional generative adversarial networks, arXiv preprint arXiv:1511.06434.
[67] A. Creswell, A.A. Bharath, Inverting the generator of a generative adversarial network, IEEE Transactions on Neural Networks and Learning Systems.
[68] J. Zhu, T. Park, P. Isola, A.A. Efros, Unpaired image-to-image translation using cycle-consistent adversarial networks, arXiv preprint.
[69] A. Krizhevsky, G. Hinton, Learning multiple layers of features from tiny images, Tech. rep., Citeseer (2009)..

Wendi Li received the B. Eng. degree in computer science from South China University of Technology, Guangzhou, China. He is currently a research intern at Microsoft Research Asia, Beijing, China. His current research interests include machine learning, evolutionary computation, and computer vision.

Wing W. Y. Ng received the B.Sc. and Ph.D. degrees from Hong Kong Polytechnic University, Hong Kong, in 2001 and 2006, respectively. He is a Professor with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China, where he is currently the Deputy Director of the Guangdong Provincial Key Laboratory of Computational Intelligence and Cyberspace Information. His current research interests include neural networks, deep learning, smart grid, smart healthcare, smart manufacturing, and non-stationary information retrieval. Dr. Ng is currently an Associate Editor of the International Journal of Machine Learning and Cybernetics. He is the Principle Investigator of four China National Natural Science Foundation projects and a Program for New Century Excellent Talents in University from China Ministry of Education. He served as the Board of Governor for IEEE Systems, Man and Cybernetics Society in 2011 and 2013.

Ting Wang received the M.Sc. degree in computer science from Northeast Normal University, Changchun, China, in 2017. She is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China. Her current research interests include learning methods and generalization capabilities for deep neural networks, and their applications in real-world problems, such as smart healthcare and smart grid.

Marcello Pelillo received the Laurea degree (summa cum laude) in computer science from the University of Bari, Bari, Italy, in 1989. He is a Full Professor of computer science with the Ca' Foscari University of Venice, Venice, Italy, where he leads the Computer Vision and Pattern Recognition Group. He holds visiting research positions in several research institutions, including Yale University, New Haven, CT, USA; McGill University, Montreal, QC, Canada; the University of Vienna, Vienna, Austria; York University, Toronto, ON, Canada; and National ICT Australia, Sydney NSW, Australia. Mr. Pelillo serves for the Editorial Boards of the journals the IEEE Transactions on Pattern Analysis and Machine Intelligence, Pattern Recognition, IET Computer Vision, Brain Informatics, Frontiers in Computer Image Analysis, and serves on the Advisory Board for the International Journal of Machine Learning and Cybernetics. He has been the General Chair for ICCV 2017, and is serving as the Program Chair for ICPR 2020. He is a fellow of the IAPR.

Sam Kwong received the B.Sc. degree in electrical engineering from the State University of New York, New York, NY, USA, in 1983, the M.Sc. degree in electrical engineering from the University of Waterloo, Waterloo, ON, Canada, in 1985, and the Ph.D. degree from Fernuniversitaet, Hagen, Germany, in 1996. From 1985 to 1987, he was a Diagnostic Engineer with the Control Data Canada, Mississauga, ON, Canada. He joined Bell Northern Research Canada, Ottawa, ON, Canada, as a Member of Scientific Staff. He joined the Department of Electronic Engineering, City University of Hong Kong, Hong Kong, as a Lecturer in 1990, where he is currently a Chair Professor with the Department of Computer Science. His current research interests include video coding, pattern recognition, and evolutionary algorithms. Prof. Kwong is currently the Vice-President of Cybernetics with the IEEE Systems, Man and Cybernetics. He also serves as an Associate Editor for the IEEE Transactions on Evolutionary Computation, the IEEE Transactions on Industrial Electronics, and the IEEE Transactions on Industrial Informatics, and the Journal of Information Science. He is also appointed as an IEEE Distinguished Lecturer of the IEEE SMC Society in 2017. He was elevated to an IEEE fellow for his contributions on optimization techniques for cybernetics and video coding in 2014.