Accepted Manuscript

An evolutionary hyper-heuristic to optimise deep belief networks for image reconstruction

Nasser R. Sabar, Ayad Turkey, Andy Song, Abdul Sattar

PII: S1568-4946(19)30280-7
DOI: <https://doi.org/10.1016/j.asoc.2019.105510>
Article number: 105510
Reference: ASOC 105510

To appear in: *Applied Soft Computing Journal*

Received date : 19 September 2017
Revised date : 25 April 2019
Accepted date : 16 May 2019

Please cite this article as: N.R. Sabar, A. Turkey, A. Song et al., An evolutionary hyper-heuristic to optimise deep belief networks for image reconstruction, *Applied Soft Computing Journal* (2019), <https://doi.org/10.1016/j.asoc.2019.105510>

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

- We propose a hyper-heuristic to optimise Deep Belief Networks.
- We use a large set of heuristic with in the hyper-heuristic.
- We propose a non-parametric statistical test to identify the best heuristics.
- We tested the proposed hyper-heuristics on three different datasets.
- The results demonstrated the effectiveness of the hyper-heuristic.

An Evolutionary Hyper-Heuristic to Optimise Deep Belief Networks for Image Reconstruction

Nasser R. Sabar ¹, Ayad Turkey ², Andy Song ², Abdul Sattar ³

¹ *Department of Computer Science and Information Technology, La Trobe University, Melbourne, Australia*

² *School of Computer Science and I.T., RMIT University, Melbourne, Australia*

³ *Institute for Integrated and Intelligent Systems (IIIS), Griffith University, Brisbane, Australia*

Abstract

Deep Belief Networks (DBN) have become a powerful tools to deal with a wide range of applications. On complex tasks like image reconstruction, DBN's performance is highly sensitive to parameter settings. Manually trying out different parameters is tedious and time consuming however often required in practice as there are not many better options. This work proposes an evolutionary hyper-heuristic framework for automatic parameter optimisation of DBN. The hyper-heuristic framework introduced here is the first of its kind in this domain. It involves a high level strategy and a pool of evolutionary operators such as crossover and mutation to generates DBN parameter settings by perturbing or modifying the current setting of a DBN. Providing a large set of operators could be beneficial to form a more effective high level strategy, but in the same time would increase the search space hence make it more difficulty to form a good strategy. To address this issue, a non-parametric statistical test is introduced to identify a subset of effective operators for different phases of the hyper-heuristic search. Three well-known image reconstruction datasets were used to evaluate the performance of the proposed framework. The results reveal that the proposed hyper-heuristic framework is very competitive when compared to the state of art methods.

Email addresses: n.sabar@latrobe.edu.au (Nasser R. Sabar ¹), ayad@rashid@gmail.com (Ayad Turkey ²), andy.song@rmit.edu.au (Andy Song ²), a.sattar@griffith.edu.au (Abdul Sattar ³)

Keywords: Hyper-heuristics, Meta-heuristics, Deep Belief Networks, Hyper-parameters, Optimisation

1. Introduction

Deep learning (DL) approaches have shown to be very powerful in dealing with complex real world problems [4] such as winning of AlphaGo over human world champions [27]. DL is also achieved unprecedented levels of accuracy for image classification and reconstruction tasks [8], [12]. However, DL training and success are highly affected by the selected parameter settings. Thus, to achieve peak performance, the user needs to properly fine-tune all DL parameters and selects the most appropriate values. This tuning process is essential but it requires a human expertise and known to be computationally expensive. This is because DL becomes more complicated with more parameter, and thus the burden of this tuning process turn to be very heavy or impractical if the data keep change from time to time. Hence, an automatic DL parameter tuning is an effective and highly desirable way to find the appropriate parameter values and their optimal combination.

To this end, an effective framework concerning the DL parameter settings and their combination has been proposed in this paper. We integrated the proposed framework with a well known variant of DL known as Deep Belief Networks (DBN). It should be noted that the same framework can be easily integrated with other DL variants. DBN was proposed by Hinton et al [11]. DBN involves a set of simple unsupervised networks. Each DBN the hidden layer of the first layer is used as input for the next layer. A widely used DBN is the one that uses Restricted Boltzmann Machines (RBM) in which a multi-layer is created by “stacking” several RBM. However, training RBM requires a domain expert and known as a difficult task as mentioned in [9] “*This requires a certain amount of practical experience to decide how to set the values of numerical meta-parameters such as the learning rate, the momentum, the weight-cost, the sparsity target, the initial values of the weights, the number of hidden units and the size of each mini-batch*” [9]. Consequently, several fine-tuning techniques have been designed for tuning DBN parameters. Example of these tuning techniques are particle swarm optimisation (PSO) [13], genetic algorithm (GA) [14], and variant of evolutionary strategies [16], [17].

An effective way to automatically fine-tune DBN parameters is by using evolutionary algorithm (EA). The major asset of EA is that it operates on

a set (population) of solutions that can be distributed over the parameter search spaces to capture the changes of the data as well as finding a high level of accuracy. Another benefit is that EA allows the transfer of problem-specific knowledge from one generation to another, which can be very helpful to deal with dynamic changes in the problem search space. Therefore, the aim of this work is to propose an evolutionary hyper-heuristic (HH) framework to automatically fine-tune DBN parameters for image reconstruction problem. The proposed HH make uses of various evolutionary operators to find the most suitable parameters in an adaptive manner. It automatically adjusts the parameter values to suit with the current problem data as well as the environmental changes. To effectively deal with various DL parameters, the proposed HH uses a large set of evolutionary operators that operate immediately on the search space of DBN parameters. An on-line non-parametric statistical test was designed to characterize the proper subset of operators from the given set as the search progresses. The proposed HH has been evaluated using three different datasets. The obtained results were compared with the state of the art methods.

The rest of this paper is structured as follows: the next section (Section II) discusses the background and related work; the proposed methodology is presented in Section III; the details on experimental settings and the name of existing methods are provided in Section IV; the results comparison is described in Section V. Section VI provides some conclusions.

2. Background and Related Work

In this section, we first discuss the deep belief networks (DBN), followed by the related work and the motivation of this work.

2.1. Deep Networks

This section consists of three subsections. The first one presents the background of Restricted Boltzmann Machines (RBM). The second subsection describes the learning method. The final subsection presents deep belief networks (DBN).

2.1.1. Restricted Boltzmann Machine

The restricted Boltzmann machines (RBM) is a generative neural network consists of two layers network. It uses energy to learn the probability distribution of the given input data. In RBM, the first (input) layer is connected

to the second (hidden) layer and the energy of the network is calculated as follows:

$$E(vv, hh) = - \sum_{i \in input} aa_i vv_i - \sum_{j \in hid} bb_j hh_j - \sum_{i,j} v_i h_j ww_{ij} \quad (1)$$

where vv_i and hh_j are the state of i^{th} node. vv is the set of nodes in the input layer. aa_i and bb_j are the biases for input layer and hidden layer. hh is the set of nodes in the hidden layer. ww_{ij} represents the weight of edge connecting two nodes. The probability of all nodes are update by the following energy formula.

$$P(vv, hh) = \frac{e^{-E(vv, hh)}}{\sum_{v, h} e^{-E(vv, hh)}} \quad (2)$$

2.1.2. Learning

The main role of learning process is to create a classification or reconstruction model by generating a partition model. This is accomplished via modifying the biases and weights of all nodes. The probability is updated as follows:

$$F(vv) = \frac{\sum_{hh} e^{-E(vv, hh)}}{\sum_{vv, hh} e^{-E(vv, hh)}} \quad (3)$$

The log probability is calculated using Equation (4). The weights and biases are then updated based on the calculated log values.

$$\frac{\partial \log P(vv)}{\partial ww_{ij}} = E[vv_i hh_j]_{dat} - E[vv_i hh_j]_{mod} \quad (4)$$

aa_i and bb_j are calculated in a similar way.

$$\frac{\partial \log P(vv)}{\partial aa_i} = vv_i - E[vv_i]_{mod} \quad (5)$$

$$\frac{\partial \log P(vv)}{\partial bb_j} = E[hh_j]_{mod} - E[hh_j]_{mod} \quad (6)$$

where E represents the distribution expectations. $E[\cdot]_{dat}$ is probabilities of data. $E[\cdot]_{mod}$ is probability of the generated model. The log probability learning can be performed using a very simple stochastic steepest ascent learning rule as follows:

$$\Delta ww_{ij} = \epsilon \epsilon (E[vv_i hh_j]_{dat} - E[vv_i hh_j]_{mod}) \quad (7)$$

$\epsilon \epsilon$ represents the learning rate and $E[vv_i hh_j]_{dat}$ is updated as follows:

$$E[vvhh]_{dat} = P(hh|vv)vv^T T \quad (8)$$

The vv , the binary state, hh_j and i , are set as follows:

$$P(hh_j = 1|vv) = \sigma(hb_j + \sum_{i=1}^k vv_i ww_{ij}) \quad (9)$$

where $\sigma(x_n)$ is the sigmoid function.

$$\sigma(x_n) = \frac{1}{1 + e^{-x_n}} \quad (10)$$

The $P(vv_i = 1|hh)$ is calculated as follows:

$$P(vv_i = 1|hh) = \sigma(aa_i + \sum_{j=1}^n vv_j ww_{ij}) \quad (11)$$

RBM uses Equation (9) and Equation (11) respectively to update all nodes of hidden and visible layers at every learning iteration. $E[vvhh]_{mod}$ is updated over a fixed number (k) of iterations. The Contrastive Divergence (CD) algorithm proposed by Hinton [10] can be used to update $E[vvhh]_{mod}$ as follows:

$$W^{s+1} = W^s + \Delta w \quad (12)$$

Equation (7) is used to update Δw . W represents the weight matrix. W^s represents the matrix at time s . The biases of all nodes are updated as follows:

$$aa^{s+1} = aa^s + \epsilon(vv - E[vv]_{mod}) \quad (13)$$

$$bb^{s+1} = bb^s + \epsilon(E[hh]_{dat} - E[hh]_{mod}) \quad (14)$$

2.1.3. Deep Belief Networks

Deep belief network (DBN) is a variant of deep neural network which involves sequence of layers of graphical model having both directed edges and undirected edges. RBM serves as a building blocks for DBN in which DBN can be created by stacking a set of RBMs on each other. The input nodes in DBN are directly connected to the nodes of the hidden layer. The output of first layer is used as an input for the second layer, the output of second layer is used as an input of the third layer and so on. RBMs are often trained using CD described above. A learning algorithm (Back propagation) can be used to update the developed network.

2.2. Related work

The state-of-the-art DBN approaches have been manually designed and tuned by a human expert. However, despite these remarkable successes, there is no guarantee that they will deliver the same performance on different applications. To this end, various techniques have been proposed in the literature to automate the design of DBN approaches. Most of these works focused on Convolutional neural networks (CNNs), a well-known variant of deep neural networks. CNNs use the DBN or stacked auto-encoder approaches to learn the meaningful representations for machine learning tasks. Approaches to tune CNNs can be classified into heuristic based-approaches and non-heuristic approaches. The non-heuristic approaches are based on the reinforcement learning methods [31] such as neural network based reinforcement learning [39], meta-modelling [3], network transformation [6] and Q-learning [38]. Heuristic based-approaches utilise evolutionary computation (EC) algorithms such as genetic algorithm, differential evolution and particle swarm optimisation. Sun et al. [30] proposed an effective and efficient approach to find the appropriate architecture and the parameter values for deep neural networks. The authors applied genetic algorithms to evolve various parameter values. [28] proposed genetic algorithm to tune CNN architectures only. [29] proposed particle swarm optimisation to evolve convolutional autoencoders for image classification. [34] extended [29] work by proposing a

variable-length particle swarm optimisation. [35] presented a hybrid differential evolution algorithm for image classification. [36] proposed a genetic CNN that uses a fixed-length binary string to encode the connections of CNN architectures.

In summary, automate the design of CNNs is a promising research direction. However, most of existing techniques are either specifically designed or have a few parameters that need to set in advance. To this end, we propose an evolutionary hyper-heuristic framework to evolve DBNs. Hyper-heuristic is a problem independent and can be applied to various application without any modifications. It also does not have too many parameters that need to be tuned in advance.

2.3. Motivation

Although several approaches have been proposed to automate the design of deep neural network variants, most of them have been specifically customised to tune a practical variant such as CNN or DBN. In addition, they use a problem specific knowledge and have few parameters that have a crucial impact on the tuning process. These issues serve as the main motivation of this work to propose an evolutionary hyper-heuristic framework that is problem independent and did not require parameters tuning an advance. Hyper-heuristic has proven to be an effective and efficient approach in dealing with various complex optimisation problems, but has not fully explored in evolving deep learning methods.

3. Methodology

This work introduces an evolutionary hyper-heuristic (HH) framework to optimise DBNs parameters (number of hidden units, learning rate, penalty parameter and weight decay). HH is a high level automatic selection method that adapt to the optimisation problem by continually selecting which operator to be applied at each decision point [5]. HH is a problem independent and it seeks for the suitable operator rather than solutions for the given problem. It has shown a remarkable success on various hard problems such as timetabling [20], [23], large scale vehicle routing problems [24], [26], rough set attribute reduction [2] and combinatorial optimisation problems [18], [19], [22]. A classification of HH frameworks is presented in [5].

3.1. The proposed framework

In this section, the proposed framework is presented. Our HH is an iterative selection framework. A traditional selection HH framework contains two levels: *high level strategy* and *low level heuristics*. The former has a selection strategy to select an operator from the given pool and an acceptance criterion to whether accept or reject the new solution. The later involves a pool of operators (denoted as *LLH*), the initial solution and the objective function (fitness or cost function). The flowchart of a typical HH framework is depicted in Figure 1.

Figure 1: A traditional selection hyper-heuristic framework

The HH works as follows. It first creates a starting solution, S_{st} (the set of DBN parameters to optimise). Next it calls the high level strategy to choose an operator to modify S_{st} so a new DBN parameter values are generated, S_{new} . The HH then calls the objective function to calculate the quality of S_{new} and the acceptance criterion to either accept or reject S_{new} . If S_{new} is accepted, DBN parameters will be updated and S_{new} will be used as a new starting solution ($S_{st}=S_{new}$). The process of selecting operator, creating a new solution, calling objective function and updating, are executed till reaching the defined stopping condition.

In the following subsection, we discuss the main parts of our selection HH framework.

3.2. The high level strategy

The high level strategy is a key component in HH and has a vital impact on HH performance [22]. Thus, designing a good high level strategy can improve HH ability in choosing the appropriate operator from the low level heuristics. A traditional high level strategy involves operator selection strategy and the acceptance criterion.

3.2.1. Selection strategy

The selection strategy manages the operator selection process. At each iteration, it decides which operator should be applied. Multi-Armed Bandit (MAB) is used as our selection strategy with the proposed HH [19]. MAB is an on-line operator selection strategy that saves the past performance of all operators (LLH). It then uses the saved performance to determine which operator (LLH) should be applied at each decision point. MAB records for each operator two variables: q_i and n_i . q_i records the average rewards obtained by the i^{th} operator up to current time t where the higher q_i the better performance of operator. n_i represents how many times the i^{th} operator has been selected where the higher the better. MAB uses q_i and n_i of all operators (LLHs) as inputs for Equation (15) to calculate the rank value all operators. The operator with maximum rank value will be selected.

$$\arg \max_{i=LLH_1, \dots, LLH_n} \left(q_i(t) + c_e \sqrt{\frac{2 \log \sum_{i=LLH_1}^{LLH_n} n_i(t)}{n_i(t)}} \right) \quad (15)$$

where $\{LLH_1, \dots, LLH_n\}$ represent the pool of operators. t represents the number of performed iterations. c_e is a constants factor controls the influence of q_i and n_i . n represents the total number of operators (LLHs) in the pool. We update $q_i(t)$ as follows:

$$q_i(t) = \frac{n_{i(t-1)} \times q_{i(t-1)} + r_{i(t)}}{n_{i(t)}} \quad (16)$$

where $r_{i(t)}$ is the total reward of LLH_i operator up to iteration t . In this work, we use Equation 17 to calculate $r_{i(t)}$.

$$r_{i(t)} = \sum \Delta f_i \quad (17)$$

In this work, $r_{i(t)}$ represents the summation of fitness improvement obtained by operator i up to iteration t .

3.2.2. Acceptance Criterion

The acceptance criterion is the second important component in the HH high level strategy. It decides whether a new solution should be accepted and used as a new starting solution or should be discarded. In this paper, we utilise Monte Carlo (MC) acceptance criterion within our HH framework [1, 21, 25]. In MC, the generated new solution will be accepted if it improves the objective evaluation value or satisfying the following condition:

$$R_r < \exp(\Delta f) = \exp(f1_t - f2_{t-1}) \quad (18)$$

where R_r generates a uniform number within $[0,1]$ interval. $\Delta f = f1_t - f2_{t-1}$.

3.3. Low level Heuristics

The performance of HH depends on the pool of operators at the low level. This work uses a large number of evolutionary operators (crossover and mutation) as our pool of operators. A pool that consist of a large set of diverse operators can be more beneficial than a small set of simple operators but it could lead to more difficulties in forming better high level strategy. In the following subsection, we discuss the pool of operators as well as our proposed method for identifying a good subset of operators to overcome the aforementioned difficulties caused by large operator set.

3.3.1. Pool of operators

In this work, we use a large group of evolutionary operators as our pool of operators. The pool involves forty different evolutionary operators (crossover and mutation), which can modify one, two, three or all parameters. The utilised operators have various characteristics and shown to perform well for various problem or different instances of the same problems. The forty operators are generated by the following ten mechanisms. Each one is used in four different ways based on the associated index. For example, *Parameterised Gaussian Mutation* has four forms: PGM_1 to PGM_4 . They apply parametrised Gaussian mutation one parameter, two parameters, three and four respectively. Thus, each basic mechanism has four different variations of operators. Below are the details.

1. Parametrised Gaussian Mutation

$$X_{new} = X_i + \mathcal{N}(Mean, \sigma^2) \quad (19)$$

where *Mean* is fixed to 0, and the standard deviation (σ_2) is fixed to 0.5

2. same as Equation (19) but σ_2 is fixed to 0.4
3. same as Equation (19) but σ_2 is fixed to 0.3
4. same as Equation (19) but σ_2 is fixed to 0.2
5. same as Equation (19) but *Mean* is fixed to 1 and σ_2 is fixed to 1

6. Differential Mutation 1

$$X_{new} = X_i + F \times (X_{1i} - X_{2i}) \quad \forall i = 1 \dots n \quad (20)$$

7. Differential Mutation 2

$$X_{new} = X_i + F \times (X_i - X_{1i}) + F \times (X_{2i} - X_{3i}) \quad (21)$$

8. Differential Mutation 3

$$X_{new} = X_i + F \times (X_{1i} - X_{2i}) + F \times (X_{3i} - X_{4i}) \quad (22)$$

In Operators 6, 7 and 8 X_i is the given solution. X_{1i} is the found best solution. X_{2i} is randomly generated within the maximum and minimum values of all DBN parameters. X_{3i} is perturbed solution created by modifying the best solution. X_{4i} is perturbed solution created by modifying X_i . F is the differential weight ($F=0.9$).

9. Arithmetic Crossover

$$X_{new} = \lambda \times X_i + (1 - \lambda) \times X_{1i} \quad (23)$$

where λ is randomly generated number within $[0, 1]$. X_i is the given solution. X_{1i} is the best solution till the current point.

10. Polynomial Mutation

$$X_{new} = \begin{cases} X_i + \sigma \times (lu - up), & \text{if } R \leq 0.5 \\ X_i, & \text{otherwise} \end{cases}$$

and

$$\sigma = \begin{cases} (2 \times R)^{\frac{1}{(\eta\eta+1)}} - 1, & \text{if } R \leq 0.5 \\ 1 - (2 - 2 \times R)^{\frac{1}{(\eta\eta+1)}}, & \text{Otherwise} \end{cases}$$

where up represents the upper value of i^{th} parameter. lu represents lower value of i^{th} parameter. $\eta\eta = 0.3$

3.3.2. Identifying the Subset of Good Operators

This work proposes a non-parametric statistical test to select the best top ten operators to form the “good” subset. We utilise the Page–Hinkley (PH) as our non-parametric statistical test to form a new subset of operators if the current subset is not performing well. PH monitors the change in quality as follows:

$$m_t = \sum_{j=1}^t e_j \quad (24)$$

$$e_t = r_t - \alpha_t + \delta \quad (25)$$

$$\alpha_t = \frac{1}{t} \sum_{j=1}^t r_j, \quad (26)$$

where α_t is the accumulated average rewards of a subset up to t . r_t is the reward of the i^{th} subset. e_t is the difference between r_t and α_t plus a constant parameter δ . m_t is the accumulated difference of e_t up to t . r is the impact of i^{th} subset at step j . PH will be activated if the difference is greater than γ ($\max_{i=1, \dots, n} \{|m_i|\} - |m_t| > \gamma$). When PH is activated, HH will randomly form a new subset.

3.4. Solution Representation

The proposed framework uses one-dimensional array to represent DBN parameters as shown in Figure (2). The size of array is set to 4 same as the total number of DBN parameters.

<i>Parameter</i>	learning rate	weight decay	penalty	# of hidden units
<i>Value</i>	0.5	0.42	0.72	3

Figure 2: Solution representation

3.5. Initial Solution

In this work, the initial (starting) solution is randomly created as follows:

$$x_j = lu_j + R_j(0, 1) \times (up_j - lu_j), \quad i = 1 \dots M \quad (27)$$

where j ($j \in 1 \dots 4$) represents the parameter index. M represents the total number of tunable parameters. R_j generates a random value between "0" and "1". lu_j is the lower value of j -th parameter. up_j is the upper bound of j -th parameter. Please note that for integer parameter, x_j is rounded to nearest integer value within the pre-defined upper and lower bounds.

3.6. Objective Function

The objective function evaluates the quality of the current solution against other one. It measure how good this solution against the problem requirement. The mean squared error (M_{error}) is utilised as our objective function within the proposed HH to calculate the error as follows:

$$M_{error} = \frac{1}{t_n} \sum_{i=1}^{t_n} (I_i - I_{original})^2 \quad (28)$$

where t_n represents total the number of training set. I_i and $I_{original}$ are the accuracy of the reconstructed and original images, respectively. In our work, the proposed framework attempts to evolve the appropriate set of DBN parameter values that minimises the minimum squared error (MSE) of the reconstructed images from the training set.

4. Experimental Set-up

The benchmark datasets that were utilised in the experiment are first described in this section. Next, we discuss the parameter settings of the proposed HH. Finally, we present the names and acronyms of existing methods that were used in the comparison.

Figure 3: Samples of Image Reconstruction Dataset

4.1. Image Dataset

In this work, the binary image reconstruction datasets used in [16] and [17] are used to assess the performance of our HH framework. Image reconstruction concerns with eliminating background noise and enhancing image to get a better classification accuracy. Image reconstruction is a challenging problem that can significantly affect the performance of the machine learning methods. The type, size and noise are the main difficulties of performing image reconstruction.

Figure 3 shows sample of images from these datasets (denoted as Sample 1: MNIST, Sample 2: CalTech 101 Silhouettes and Sample 3: Semeion Handwritten Digits).

1. MNIST

MNIST dataset involves handwritten digits images ('0' to '9') in gray scale. All image are 28×28 resolution. We use 1,200 images in the training phase and 10,000 images in the testing phase.

2. CalTech 101 Silhouettes

CalTech 101 Silhouettes involves 101 high quality polygon images. We use 28×28 resolution in training and testing phases.

3. Semeion Handwritten Digits

Semeion handwritten digits is handwritten digits images ('0' to '9'). It differs from the above samples in the writing way. It has two different ways: calmly and accurately (normal), and fast with no attention (hasty fast). 16×16 resolution were used for all images.

4.2. DBN Parameters

Table 1 shows DBN parameters along with the range of each parameter. We have tested a range of values for most of the parameters to test the suitability. The best values that lead to a reasonable trade-off between solution quality and the computational time needed to reach good quality solutions were selected. The ranges are chosen to facilitate the comparison so these test cases are consistent with other studies.

Table 1: The tested parameter values

#	Parameter name	Tested value
1-	Weight decay	0.1 - 0.9
2-	Learning rate	0.1 - 0.9
3-	Batches-size	20
4-	Penalty	0.00001 - 0.01
5-	No. of hidden	5 - 100
6-	No. of layers	1, 2, 3
7-	No. of epochs	10
8-	No. of generations	100

4.3. DBN tuning methods

The proposed HH framework for DBN parameters is compared with four existing methods. Table 2 shows the acronyms and references of these methods. They are denoted as PSO, ES, IES and FFA in this work. These methods are reported in recent papers [17],[16] and we have followed the same experimental settings.

Table 2: Acronyms of the compared methods

Acronym	Name	References
FFA	FireFly algorithm	[37]
ES	Variant of Evolution Strategies	[7]
PSO	Particle Swarm Optimisation	[13]
IES	Improved Evolution Strategies	[15]

To ensure a fair comparison, all methods (HH, FFA, ES, PSO and IES) used the same experimental procedures as in [17],[16]. We have used 10-fold cross validation which randomly divided the data into training and testing. The evolved set of parameter values is used to reconstruct the images of the test set. We have used three DBN models (denoted as L_1 , L_2 and L_3) as follows:

- L_1 : uses one layer of RBM.
- L_2 : uses two layer of RBM.
- L_3 : uses three layer of RBM.

We have used three different algorithms (denoted as A_1 , A_2 and A_3) to train these DBN models.

- A_1 : Contrastive Divergence [10].
- A_2 : Persistent Contrastive Divergence [32].
- A_3 : Fast Persistent Contrastive Divergence [33].

5. Results and Comparisons

This section first investigates the impact of the designed components on HH performance and then compares the obtained results against FFA, ES, PSO and IES.

5.1. Effectiveness Evaluation

To assess the benefit of the components integrated within proposed HH framework, three benchmark datasets were used. The following HH variants and algorithm are used for comparison.

- *HH*: the proposed HH that uses a large set of operators and PH test for forming the heuristic subset.
- *HH₁*: uses a large set of operators - the forty heuristics but without PH.
- *HH₂*: uses a small set of operators - the ten basic operators.
- *GD*: gradient descent algorithm.

Table 3: The results of HH compared to *HH₁* and *HH₂*

	MNIST	Caltech 101	Semeion
<i>HH</i>	0.0861	0.1572	0.2055
<i>HH₁</i>	0.0867	0.158	0.2059
<i>HH₂</i>	0.0871	0.1582	0.2062
<i>GD</i>	0.0895	0.1610	0.2811

The above variants (*HH*, *HH₁*, *HH₂* and *GD*) were tested using the same computational resources. To ensure a fair comparison, A_1 was used with all tested (*HH*, *HH₁*, *HH₂* and *GD*). We have fixed the number of layers into two (L_2). All variants (*HH*, *HH₁*, *HH₂* and *GD*) were executed 20 times and the obtained MSE over 20 runs are reported in Table 3. We have used 10-fold cross validation which randomly divided the data into training and testing. In our work, the evolved set of parameter values is used to reconstruct the images on the test set. The best obtained result on each dataset is showed in bold in Table 3.

From Table 3 one can clearly see that, on all the image dataset, *HH* is outperformed *HH₁*, *HH₂* and *GD*. The results also demonstrate that *HH₁* is better than *HH₂* and *GD* is the worse performing method. That verifies the benefit of using large set of operators. The bad performance of *GD* is due to the fact that *GD* is often very fast gets stuck in local optima point which

Table 4: The p -values of HH versus HH_1 and HH_2

HH vs.	MNIST	Caltech 101	Semeion
HH_1	0.040	0.036	0.041
HH_2	0.000	0.000	0.000
GD	0.000	0.000	0.000

prevent the search from exploring different areas in the parameter search spaces.

To further examine this positive outcome, a wilcoxon pairwise statistical comparison test 0.05 confidence interval was performed. In Table 4, we list the p -values of HH versus HH_1 , HH_2 and GD where a p -value less than 0.05 indicates HH is significantly better than other method. Through this statistical analysis presented in Table 4, we can observe that all p -values of HH versus other variants are lower than 0.05, indicating the better performance of HH is indeed significant.

5.2. Comparison with State of the Art Methods

The results obtained by our proposed HH for the three dataset (MNIST, Caltech 101 Silhouettes and Semeion Handwritten Digits) are presented and compared with FFA, LS, PSO and IES in Tables 5, 6, 7, respectively. All tables have been divided into three sections. The first section contains results for the one layer (L_1) DBN model, the second one for the two layers (L_2) DBN model and the third section for the three layers (L_3) DBN model. Each section has three columns: the first one contains obtained MSE using A_1 , the second one represents the MSE achieved by of A_2 and third one represents the MSE produced of A_3 . That is we have trained DBN each model (L_1 , L_2 or L_3) three times using A_1 , A_2 and A_3 on all tested dataset (MNIST, Caltech 101 Silhouettes and Semeion Handwritten Digits). The reported MSEs are the average over 20 runs with different random seeds. In these table, the best MSE (the lower the better) produced by the tested algorithm and DBN model is marked in boldfont.

From the tables, one can clearly see that the proposed HH framework obtained the smallest MSE errors on all tested datasets for all DBN models (L_1 , L_2 and L_3) using the three learning algorithms (A_1 , A_2 and A_3). That means the good performance of the proposed HH is not dependent on training

algorithms and pre-defined models. Such results provide strong support that HH is an effective method for optimising of DBN parameters.

Table 5: The obtained MSE for MNIST dataset

Method	L_1			L_2			L_3		
	A_1	A_2	A_3	A_1	A_2	A_3	A_1	A_2	A_3
FFA	0.0876	0.0876	0.0882	0.0876	0.0876	0.0882	0.0876	0.0876	0.0885
ES	0.1059	0.1325	0.1324	0.1059	0.1061	0.1057	0.1059	0.1058	0.1057
PSO	0.1057	0.1058	0.1057	0.106	0.1059	0.1059	0.1058	1.1059	0.1058
IES	0.0903	0.0879	0.0882	0.0885	0.0886	0.0886	0.0887	0.0885	0.0886
HH	0.0861	0.0861	0.0867	0.0866	0.0870	0.0869	0.0866	0.0864	0.0863

Table 6: The obtained MSE for on Caltech 101 Silhouettes dataset

Method	L_1			L_2			L_3		
	A_1	A_2	A_3	A_1	A_2	A_3	A_1	A_2	A_3
FFA	0.1589	0.1598	0.1616	0.1603	0.1606	0.1635	0.1606	0.1606	0.1625
ES	0.1695	0.1696	0.1691	0.1695	0.1699	0.1693	0.1694	0.1696	0.1692
PSO	0.1691	0.169	0.1689	0.1685	0.1691	0.1688	0.1692	0.1692	0.169
IES	0.1696	0.1695	0.1693	0.1699	0.1607	0.1612	0.1611	0.1618	0.1606
HH	0.1580	0.1578	0.1581	0.1572	0.1568	0.1570	0.1565	0.1578	0.1574

Table 7: The obtained MSE for on Semeion Handwritten Digits dataset

Method	L_1			L_2			L_3		
	A_1	A_2	A_3	A_1	A_2	A_3	A_1	A_2	A_3
FFA	0.2068	0.2078	0.2103	0.2097	0.2096	0.2125	0.2097	0.2096	0.2115
ES	0.2128	0.2128	0.2129	0.2202	0.2128	0.2128	0.2199	0.2128	0.2128
PSO	0.2128	0.2128	0.2128	0.2128	0.2128	0.2128	0.2128	0.2128	0.2127
IES	0.2131	0.213	0.2128	0.2116	0.2114	0.2121	0.2103	0.2109	0.2119
HH	0.2056	0.2058	0.2061	0.2055	0.2062	0.2060	0.2073	0.2075	0.2066

6. Conclusion

In this paper, we have proposed an evolutionary hyper-heuristic framework to optimise deep belief networks performance for image reconstruction problems. It is the first study of its kind introducing hyper-heuristic in deep learning. The proposed framework has a high level strategy to automate the operator selection process and various low level heuristics to evolve new deep belief networks parameter settings. The low level heuristics use a large

number of problem specific evolutionary operators (crossover and mutation). In addition, a subset selection mechanism is proposed to identify good subset of operators. Three different datasets were used to assess the performance of the proposed framework. Three different deep belief network models using three different learning algorithms were tested and compared. The results demonstrate the effectiveness of the proposed framework in improving the performance of deep belief networks. It has also obtained better results compared to existing meta-heuristics based deep belief networks parameter optimisation methods. This paper is a part of our work on evolving deep learning by hyper-heuristics. There are a couple of extensions that are worthy of further investigation. The most obvious one is introducing heuristic construction, so the framework would be capable of creating new heuristics rather than manipulating given heuristics. In addition we will involve more image data and possible other types of data for evaluation. Another straightforward extension is the use of this framework on image classifiers. The need for parameter optimisation for image classification is equivalent to, if not greater than, that for image reconstruction. In addition, image classification and image reconstruction are similar in many aspects hence this extension is a nature step of our further work.

References

- [1] Abdullah, S., Sabar, N. R., Nazri, M. Z. A., and Ayob, M. (2014). An exponential monte-carlo algorithm for feature selection problems. *Computers & Industrial Engineering*, 67:160–167.
- [2] Abdullah, S., Sabar, N. R., Nazri, M. Z. A., Turabieh, H., and McCollum, B. (2010). A constructive hyper-heuristics for rough set attribute reduction. In *Intelligent Systems Design and Applications (ISDA), 2010 10th International Conference on*, pages 1032–1035. IEEE.
- [3] Baker, B., Gupta, O., Naik, N., and Raskar, R. (2016). Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*.
- [4] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828.

- [5] Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. R. (2010). A classification of hyper-heuristic approaches. In *Handbook of metaheuristics*, pages 449–468. Springer.
- [6] Cai, H., Chen, T., Zhang, W., Yu, Y., and Wang, J. (2018). Efficient architecture search by network transformation. In *Thirty-Second AAAI Conference on Artificial Intelligence*.
- [7] Geem, Z. W. (2009). *Music-Inspired Harmony Search Algorithm: Theory and Applications*. Springer Publishing Company, Incorporated, 1st edition.
- [8] He, K., Zhang, X., Ren, S., and Sun, J. (2013). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.
- [9] Hinton, G. (2010). A practical guide to training restricted boltzmann machines. *Momentum*, 9(1):926.
- [10] Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800.
- [11] Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554.
- [12] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.
- [13] Kuremoto, T., Kimura, S., Kobayashi, K., and Obayashi, M. (2012). Time series forecasting using restricted boltzmann machine. In *International Conference on Intelligent Computing*, pages 17–22. Springer.
- [14] Liu, K., Zhang, L. M., and Sun, Y. W. (2014). Deep boltzmann machines aided design based on genetic algorithms. In *Applied Mechanics and Materials*, volume 568, pages 848–851. Trans Tech Publ.
- [15] Mahdavi, M., Fesanghary, M., and Damangir, E. (2007). An improved harmony search algorithm for solving optimization problems. *Applied mathematics and computation*, 188(2):1567–1579.
- [16] Papa, J. P., Scheirer, W., and Cox, D. D. (2016). Fine-tuning deep belief networks using harmony search. *Applied Soft Computing*, 46:875–885.

- [17] Rosa, G., Papa, J., Costa, K., Passos, L., Pereira, C., and Yang, X.-S. (2016). Learning parameters in deep belief networks through firefly algorithm. In *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*, pages 138–149. Springer.
- [18] Sabar, N., Ayob, M., Kendall, G., and Qu, R. (2014). The automatic design of hyper-heuristic framework with gene expression programming for combinatorial optimization problems. *Evolutionary Computation, IEEE Transactions on*, PP(99):1–1.
- [19] Sabar, N., Ayob, M., Kendall, G., and Qu, R. (2015a). A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems. *Cybernetics, IEEE Transactions on*, 45(2):217–228.
- [20] Sabar, N. R. and Ayob, M. (2009). Examination timetabling using scatter search hyper-heuristic. In *Data Mining and Optimization, 2009. DMO'09. 2nd Conference on*, pages 127–131. IEEE.
- [21] Sabar, N. R., Ayob, M., and Kendall, G. (2009). Tabu exponential monte-carlo with counter heuristic for examination timetabling. In *Computational Intelligence in Scheduling, 2009. CI-Sched'09. IEEE Symposium on*, pages 90–94. IEEE.
- [22] Sabar, N. R., Ayob, M., Kendall, G., and Qu, R. (2013). Grammatical evolution hyper-heuristic for combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation*, 17(6):840–861.
- [23] Sabar, N. R., Ayob, M., Qu, R., and Kendall, G. (2012). A graph coloring constructive hyper-heuristic for examination timetabling problems. *Applied Intelligence*, 37(1):1–11.
- [24] Sabar, N. R. and Kendall, G. (2014). Population based monte carlo tree search hyper heuristic for combinatorial optimization problems. *Information Sciences*.
- [25] Sabar, N. R., Kendall, G., and Ayob, M. (2014b). An exponential monte-carlo local search algorithm for the berth allocation problem. In *10th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2010), York, UK*, pages 544–548.

- [26] Sabar, N. R., Zhang, X. J., and Song, A. (2015b). A math-hyper-heuristic approach for large-scale vehicle routing problems with time windows. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pages 830–837. IEEE.
- [27] Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503.
- [28] Sun, Y., Xue, B., Zhang, M., and Yen, G. G. (2018a). Automatically designing cnn architectures using genetic algorithm for image classification. *arXiv preprint arXiv:1808.03818*.
- [29] Sun, Y., Xue, B., Zhang, M., and Yen, G. G. (2018b). A particle swarm optimization-based flexible convolutional autoencoder for image classification. *IEEE transactions on neural networks and learning systems*.
- [30] Sun, Y., Yen, G. G., and Yi, Z. (2019). Evolving unsupervised deep neural networks for learning meaningful representations. *IEEE Transactions on Evolutionary Computation*, 23(1):89–103.
- [31] Sutton, R. S., Parto, A. G., et al. (1998). *Introduction to reinforcement learning*, volume 135. MIT press Cambridge.
- [32] Tieleman, T. (2008). Training restricted boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th Int’l conf. on Machine Learning*, pages 1064–1071. ACM.
- [33] Tieleman, T. and Hinton, G. (2009). Using fast weights to improve persistent contrastive divergence. In *Proceedings of the 26th Int’l conf. on Machine Learning*, pages 1033–1040. ACM.
- [34] Wang, B., Sun, Y., Xue, B., and Zhang, M. (2018a). Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE.

- [35] Wang, B., Sun, Y., Xue, B., and Zhang, M. (2018b). A hybrid differential evolution approach to designing deep convolutional neural networks for image classification. In *Australasian Joint Conference on Artificial Intelligence*, pages 237–250. Springer.
- [36] Xie, L. and Yuille, A. (2017). Genetic cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1379–1388.
- [37] Yang, X.-S. (2010). Firefly algorithm, stochastic test functions and design optimisation. *International Journal of Bio-Inspired Computation*, 2(2):78–84.
- [38] Zhong, Z., Yan, J., and Liu, C.-F. (2021). Practical network blocks design with q-learning. *arXiv preprint arXiv:1708.05552*, 1(2):5.
- [39] Zoph, B. and Le, Q. V. (2016). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.