Learning Parameters in Deep Belief Networks Through Firefly Algorithm

Conference Paper · July 2016

DOI: 10.1007/978-3-319-46182-3_12

6 authors, including:

Gustavo H. de Rosa
São Paulo State University
30 PUBLICATIONS **173** CITATIONS
SEE PROFILE

João Paulo Papa
São Paulo State University
308 PUBLICATIONS **3,177** CITATIONS
SEE PROFILE

Kelton A. P. da Costa
University of São Paulo
39 PUBLICATIONS **265** CITATIONS
SEE PROFILE

Leandro Aparecido Passos Júnior
São Paulo State University
25 PUBLICATIONS **69** CITATIONS
SEE PROFILE

Learning Parameters in Deep Belief Networks Through Firefly Algorithm

Gustavo Rosa[1], João Papa[1], Kelton Costa[1],
Leandro Passos[2], Clayton Pereira[2], and Xin-She Yang[3]

[1] Department of Computing, São Paulo State University
gth.rosa@uol.com.br,kelton.costa@gmail.com
papa@fc.unesp.br
[2] Department of Computing, Federal University of São Carlos
{leandropassosjr,claytontey}@gmail.com
[3] School of Science and Technology, Middlesex University
x.yang@mdx.ac.uk

Abstract. Restricted Boltzmann Machines (RBMs) are among the most widely pursed techniques in the context of deep learning-based applications. Their usage enables sundry parallel implementations, which have become pivotal in nowadays large-scale-oriented applications. In this paper, we propose to address the main shortcoming of such models, i.e. how to properly fine-tune their parameters, by means of the Firefly Algorithm, as well as we also consider Deep Belief Networks, a stacked-driven version of the RBMs. Additionally, we also take into account Harmony Search, Improved Harmony Search and the well-known Particle Swarm Optimization for comparison purposes. The results obtained showed the Firefly Algorithm is suitable to the context addressed in this paper, since it obtained the best results in all datasets.

Keywords: Deep Belief Networks; Deep Learning; Firefly Algorithm

1 Introduction

Even today, there are still some open computer vision-related problems concerning on how to create and produce good representations of the real world, such as machine learning systems that can detect and further classify objects [4]. These techniques have been paramount during the last years, since there is an increasing number of applications that require intelligent-based decision-making processes. An attractive skill of pattern recognition techniques related to deep learning has drawn a considerable amount of interest in the last years [3], since their outstanding results have settled a hallmark for several applications, such as speech, face and emotion recognition, among others.

Roughly speaking, deep learning algorithms are shaped by means of several layers of a predefined set of operations. Restricted Boltzmann Machines (RBMs), for instance, have attracted considerable focus in the last years due to their simplicity, high level of parallelism and strong representation ability [7]. RBMs can

be interpreted as stochastic neural networks, being mainly used for image reconstruction and collaborative filtering through unsupervised learning [2]. Later on, Hinton et al. [8] realized one can obtain more complex representations by stacking a few RBMs on top of each other, thus leading to the so-called Deep Belief Networks (DBNs).

One of the main loopholes of DBNs concerns with the proper calibration of their parameters. The task of fine-tuning parameters in machine learning aims at finding suitable values for that parameters in order to maximize some fitness function, such as a classifier's recognition accuracy when dealing with supervised problems, for instance. Auto-learning tools are often used to handle this problem [16], which usually combine parameter fine-tuning with feature selection techniques. On the other hand, meta-heuristic techniques are among the most used ones for optimization problems, since they provide simple and elegant solutions in a wide range of applications. Such techniques may comprehend swarm- and population-based algorithms, as well as stochastic and nature-inspired solutions.

Some years ago, Yang [20] proposed the Firefly Algorithm (FFA), which tumbles in the pitch of meta-heuristic optimization techniques. Elementarily, the idea of the Firefly Algorithm is to solve optimization problems based on the way fireflies communicate with themselves. In other words, the idea is to model the process in which fireflies attract their mates using their flashing lights, which are produced by bioluminescence processes. The rate of flashing, amount of time and the rhythmic flash are part of their signal system, helping its association with some fitness function in order to optimize it.

However, the reader may find just a few recent works that face the problem of calibrating the parameters of both RBMs and DBNs by means of meta-heuristic techniques. Huang et al. [9], for instance, employed the well-known Particle Swarm Optimization (PSO) to optimize the number of input (visible) and hidden neurons, as well as the RBM learning rate in the context of time series forecasting prediction. Later on, Liu et al. [10] applied Genetic Algorithm to optimize the parameters of Deep Boltzmann Machines. Papa et al. [13] proposed to optimize RBMs by means of Harmony Search, and Papa et al. [14] fine-tuned the parameters of a Discriminative Restricted Boltzmann Machines. Very recently, Papa et. al [15] proposed to optimize DBNs by means of Harmony Search-based techniques.

As far as we know, FFA has never been applied for DBN parameter calibration. Therefore, the main contribution of this paper is twofold: (i) to introduce FFA to the context of DBN parameter calibration for binary image reconstruction purposes, and (ii) to fill the lack of research regarding DBN parameter optimization by means of meta-heuristic techniques. The remainder of this paper is organized as follows. Sections 2 and 3 present some theoretical background with respect to DBN and FFA, respectively. The methodology is discussed in Section 4, while Section 5 presents the experimental results. Finally, Section 6 states conclusions and future works.

2 Deep Belief Networks

In this section, we describe the main concepts related to Deep Belief Networks, but with a special attention to the theoretical background of RBMs, which are the basis for DBN understanding.

2.1 Restricted Boltzmann Machines

Restricted Boltzmann Machines are energy-based stochastic neural networks composed by two layers of neurons (visible and hidden), in which the learning phase is conducted by means of an unsupervised fashion. The RBM is similar to the classical Boltzmann Machine [1], except that no connections between neurons of the same layer are allowed. The architecture of a Restricted Boltzmann Machine is comprised by a visible layer $\mathbf{v}$ with m units and a hidden layer $\mathbf{h}$ with n units. The real-valued $m \times n$ matrix $\mathbf{W}$ models the weights between visible and hidden neurons, where w_{ij} stands for the weight between the visible unit v_i and the hidden unit h_j.

Let us assume $\mathbf{v}$ and $\mathbf{h}$ as the binary visible and hidden units, respectively. In other words, $\mathbf{v} \in \{0,1\}^m$ and $\mathbf{h} \in \{0,1\}^n$. The energy function of a Bernoulli Restricted Boltzmann Machine is given by:

$$E(\mathbf{v},\mathbf{h}) = -\sum_{i=1}^{m} a_i v_i - \sum_{j=1}^{n} b_j h_j - \sum_{i=1}^{m}\sum_{j=1}^{n} v_i h_j w_{ij}, \tag{1}$$

where $\mathbf{a}$ and $\mathbf{b}$ stand for the biases of visible and hidden units, respectively. The probability of a configuration $(\mathbf{v},\mathbf{h})$ is computed as follows:

$$P(\mathbf{v},\mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{\sum\limits_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}}, \tag{2}$$

where the denominator of above equation is a normalization factor that stands for all possible configurations involving the visible and hidden units. In short, the RBM learning algorithm aims at estimating $\mathbf{W}$, $\mathbf{a}$ and $\mathbf{b}$. The next section describes in more details this procedure.

2.2 Learning Algorithm

The parameters of an RBM can be optimized by performing stochastic gradient ascent on the log-likelihood of training patterns. Given a training sample (visible unit), its probability is computed over all possible hidden units, as follows:

$$P(\mathbf{v}) = \frac{\sum\limits_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}}{\sum\limits_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}}. \tag{3}$$

In order to update the weights and biases, it is necessary to compute the following derivatives:

$$\frac{\partial \log P(\mathbf{v})}{\partial w_{ij}} = E[h_j v_i]^{data} - E[h_j v_i]^{model}, \tag{4}$$

$$\frac{\partial \log P(\mathbf{v})}{\partial a_i} = v_i - E[v_i]^{model}, \tag{5}$$

$$\frac{\partial \log P(\mathbf{v})}{\partial b_j} = E[h_j]^{data} - E[h_j]^{model}, \tag{6}$$

where $E[\cdot]$ stands for the expectation operation, and $E[\cdot]^{data}$ and $E[\cdot]^{model}$ correspond to the data-driven and the reconstructed-data-driven probabilities, respectively.

In practical terms, we can compute $E[h_j v_i]^{data}$ considering $\mathbf{h}$ and $\mathbf{v}$ as follows:

$$E[\mathbf{hv}]^{data} = P(\mathbf{h}|\mathbf{v})\mathbf{v}^T, \tag{7}$$

where $P(\mathbf{h}|\mathbf{v})$ stands for the probability of obtaining $\mathbf{h}$ given the visible vector (training data) $\mathbf{v}$:

$$P(h_j = 1|\mathbf{v}) = \sigma\left(\sum_{i=1}^{m} w_{ij} v_i + b_j\right), \tag{8}$$

where $\sigma(\cdot)$ stands for the logistic sigmoid function. Therefore, it is straightforward to compute $E[\mathbf{hv}]^{data}$: given a training data $\mathbf{x} \in \mathcal{X}$, where $\mathcal{X}$ stands for a training set, we just need to set $\mathbf{v} \leftarrow \mathbf{x}$ and then employ Equation 8 to obtain $P(\mathbf{h}|\mathbf{v})$. Further, we use Equation 7 to finally obtain $E[\mathbf{hv}]^{data}$.

The next question now is how to obtain $E[\mathbf{hv}]^{model}$, which is the model learned by the system. One possible strategy is to perform alternating Gibbs sampling starting at any random state of the visible units until a certain convergence criterion, such as k steps, for instance. The Gibbs sampling consists of updating hidden units using Equation 8 followed by updating the visible units using $P(\mathbf{v}|\mathbf{h})$, given by:

$$P(v_i = 1|\mathbf{h}) = \sigma\left(\sum_{j=1}^{n} w_{ij} h_j + a_i\right), \tag{9}$$

and then updating the hidden units once again using Equation 8. In short, it is possible to obtain an estimative of $E[\mathbf{hv}]^{model}$ by initializing the visible unit with random values and then performing Gibbs sampling, in which $E[\mathbf{hv}]^{model}$ can be approximated after k iterations. Notice a single iteration is defined by computing $P(\mathbf{h}|\mathbf{v})$, followed by computing $P(\mathbf{v}|\mathbf{h})$ and then computing $P(\mathbf{h}|\mathbf{v})$ once again.

For the sake of explanation, the Gibbs sampling employs $P(\mathbf{v}|\tilde{\mathbf{h}})$ instead of $P(\mathbf{v}|\mathbf{h})$, and $P(\tilde{\mathbf{h}}|\tilde{\mathbf{v}})$ instead of $P(\mathbf{h}|\mathbf{v})$. Essentially, they stand for the same

meaning, but $P(\mathbf{v}|\tilde{\mathbf{h}})$ is used here to denote the visible unit $\mathbf{v}$ is going to be reconstructed using $\tilde{\mathbf{h}}$, which was obtained through $P(\mathbf{h}|\mathbf{v})$. The same takes place with $P(\tilde{\mathbf{h}}|\tilde{\mathbf{v}})$, that reconstructs $\tilde{\mathbf{h}}$ using $\tilde{\mathbf{v}}$, which was obtained through $P(\mathbf{v}|\tilde{\mathbf{h}})$.

However, the procedure depicted above is time-consuming, being also quite hard to establish suitable values for k. Fortunately, Hinton [6] introduced a faster methodology to compute $E[\mathbf{hv}]^{model}$ based on contrastive divergence. Basically, the idea is to initialize the visible units with a training sample, to compute the states of the hidden units using Equation 8, and then to compute the states of the visible unit (reconstruction step) using Equation 9. Roughly speaking, this is equivalent to perform Gibbs sampling using $k = 1$.

Based on the above assumption, we can now compute $E[\mathbf{hv}]^{model}$ as follows:

$$E[\mathbf{hv}]^{model} = P(\tilde{\mathbf{h}}|\tilde{\mathbf{v}})\tilde{\mathbf{v}}^T. \tag{10}$$

Therefore, the equation below leads to a simple learning rule for updating the weight matrix $\mathbf{W}$, as follows:

$$\begin{aligned} \mathbf{W}^{t+1} &= \mathbf{W}^t + \eta(E[\mathbf{hv}]^{data} - E[\mathbf{hv}]^{model}) \\ &= \mathbf{W}^t + \eta(P(\mathbf{h}|\mathbf{v})\mathbf{v}^T - P(\tilde{\mathbf{h}}|\tilde{\mathbf{v}})\tilde{\mathbf{v}}^T), \end{aligned} \tag{11}$$

where $\mathbf{W}^t$ stands for the weight matrix at time step t, and η corresponds to the learning rate. Additionally, we have the following formulae to update the biases of the visible and hidden units:

$$\begin{aligned} \mathbf{a}^{t+1} &= \mathbf{a}^t + \eta(\mathbf{v} - E[\mathbf{v}]^{model}) \\ &= \mathbf{a}^t + \eta(\mathbf{v} - \tilde{\mathbf{v}}), \end{aligned} \tag{12}$$

and

$$\begin{aligned} \mathbf{b}^{t+1} &= \mathbf{b}^t + \eta(E[\mathbf{h}]^{data} - E[\mathbf{h}]^{model}) \\ &= \mathbf{b}^t + \eta(P(\mathbf{h}|\mathbf{v}) - P(\tilde{\mathbf{h}}|\tilde{\mathbf{v}})), \end{aligned} \tag{13}$$

where $\mathbf{a}^t$ and $\mathbf{b}^t$ stand for the visible and hidden units biases at time step t, respectively. In short, Equations 11, 12 and 13 are the vanilla formulation for updating the RBM parameters.

Later on, Hinton [7] introduced a weight decay parameter λ, which penalizes weights with large magnitude, as well as a momentum parameter α to control possible oscillations during the learning process. Therefore, we can rewrite Equations 11, 12 and 13 as follows:

$$\mathbf{W}^{t+1} = \mathbf{W}^t + \underbrace{\eta(P(\mathbf{h}|\mathbf{v})\mathbf{v}^T - P(\tilde{\mathbf{h}}|\tilde{\mathbf{v}})\tilde{\mathbf{v}}^T) - \lambda\mathbf{W}^t + \alpha\Delta\mathbf{W}^{t-1}}_{=\Delta\mathbf{W}^t}, \tag{14}$$

$$\mathbf{a}^{t+1} = \mathbf{a}^t + \underbrace{\eta(\mathbf{v} - \tilde{\mathbf{v}}) + \alpha \Delta \mathbf{a}^{t-1}}_{=\Delta \mathbf{a}^t} \tag{15}$$

and

$$\mathbf{b}^{t+1} = \mathbf{b}^t + \underbrace{\eta(P(\mathbf{h}|\mathbf{v}) - P(\tilde{\mathbf{h}}|\tilde{\mathbf{v}})) + \alpha \Delta \mathbf{b}^{t-1}}_{=\Delta \mathbf{b}^t}. \tag{16}$$

2.3 Stacked Restricted Boltzmann Machines

Truly speaking, DBNs are composed of a set of stacked RBMs, being each of them trained using the learning algorithm presented in Section 2.2 in a greedy fashion, which means an RBM at a certain layer does not consider others during its learning procedure. In this case, we have a DBN composed of L layers, being $\mathbf{W}^i$ the weight matrix of RBM at layer i. Additionally, we can observe the hidden units at layer i become the input units to the layer $i + 1$.

The approach proposed by Hinton et al. [8] for the training step of DBNs also considers a fine-tuning as a final step after the training of each RBM. Such procedure can be performed by means of a Backpropagation or Gradient descent algorithm, for instance, in order to adjust the matrices $\mathbf{W}^i$, $i = 1, 2, \ldots, L$. The optimization algorithm aims at minimizing some error measure considering the output of an additional layer placed on the top of the DBN after its former greedy training. Such layer is often composed of softmax or logistic units, or even some supervised pattern recognition technique.

3 Firefly Algorithm

The Firefly Algorithm is derived from the flash attractiveness of fireflies when mating partners (communication) and attracting potential preys [20]. Let $\mathbf{X}_{M \times N}$ be a firefly swarm, where N is the number of variables to be optimized and M stands for the number of fireflies. Also, each firefly i is associated to a position $\boldsymbol{x}_i \in \Re^N$, $i = 1, 2, \ldots, M$, and its brightness is determined by the value of the objective function at that position, i.e. $f(\boldsymbol{x}_i)$.

The attractiveness of a firefly $\boldsymbol{x}_i$ with respect to another firefly $\boldsymbol{x}_j$, i.e. $\beta(\boldsymbol{x}_i, \boldsymbol{x}_j)$, varies with the distance r_{ij} (i.e Euclidean distance) between firefly i and firefly j, which is modeled as follows:

$$\beta(\boldsymbol{x}_i, \boldsymbol{x}_j) = \beta_0 \exp(-\gamma * r_{i,j}^2), \tag{17}$$

where β_0 is the initial attractiveness and γ the light absorption coefficient.

Each firefly at position $\boldsymbol{x}_i$ and time step t is attracted towards every other brighter firefly at position $\boldsymbol{x}_j$ (i.e. $f(\boldsymbol{x}_j) > f(\boldsymbol{x}_i)$) as follows:

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \beta(\boldsymbol{x}_i^t, \boldsymbol{x}_j^t)(\boldsymbol{x}_j^t - \boldsymbol{x}_i^t) + \alpha \left(\mu \frac{1}{2}\right), \tag{18}$$

where α is a randomization factor that controls the magnitude of the stochastic perturbation of $\boldsymbol{x}_i^t$, and $\mu \in \mathcal{U}(0,1)$. Soon after all fireflies at time step t had been modified, the best firefly $\boldsymbol{x}^*$ will perform a controlled random walk across the search space:

$$\boldsymbol{x}^* = \boldsymbol{x}_i^* + \alpha \left(\psi \frac{1}{2}\right), \tag{19}$$

where $\psi \in \mathcal{U}(0,1)$.

4 Methodology

In this section, we present the proposed approach for DBN parameter calibration, as well as we describe the employed datasets and the experimental setup.

4.1 Modelling DBN Parameter Optimization

We propose to model the problem of selecting suitable parameters considering DBN in the task of binary image reconstruction. As aforementioned in Section 2.2, the learning step has four parameters: the learning rate η, weight decay λ, penalty parameter α, and the number of hidden units n. Therefore, we have a four-dimensional search space with three real-valued variables, as well as the integer-valued number of hidden units. Roughly speaking, the proposed approach aims at selecting the set of DBN parameters that minimizes the minimum squared error (MSE) of the reconstructed images from the training set. After that, the selected set of parameters is thus applied to reconstruct the images of the test set.

4.2 Datasets

We employed three datasets, as described below:

- MNIST dataset[4]: it is composed of images of handwritten digits. The original version contains a training set with $60,000$ images from digits '0'-'9', as well as a test set with $10,000$ images[5]. Due to the high computational burden for DBN model selection, we decided to employ the original test set together with a reduced version of the training set[6].
- CalTech 101 Silhouettes Data Set[7]: it is based on the former Caltech 101 dataset, and it comprises silhouettes of images from 101 classes with resolution of 28×28. We have used only the training and test sets, since our optimization model aims at minimizing the MSE error over the training set.

[4] http://yann.lecun.com/exdb/mnist/

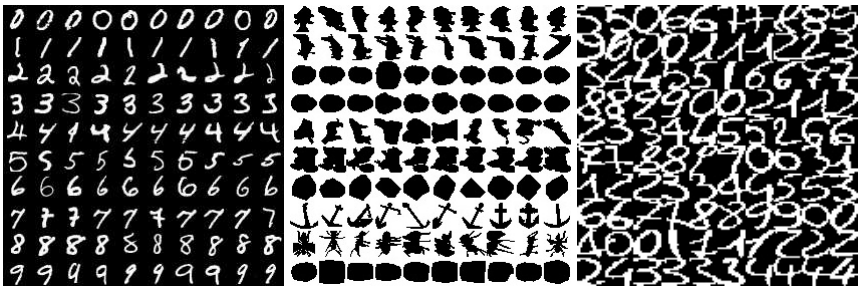
[5] The images are originally available in grayscale with resolution of 28×28.

[6] The original training set was reduced to 2% of its former size, which corresponds to $1,200$ images.

[7] https://people.cs.umass.edu/ marlin/data.shtml

– Semeion Handwritten Digit Data Set[8]: it is formed by $1,593$ images from handwritten digits '0' - '9' written in two ways: the first time in a normal way (accurately) and the second time in a fast way (no accuracy). In the end, they were stretched with resolution of 16×16 in a grayscale of 256 values and then each pixel was binarized.

Figure 1 displays some training examples from the above datasets.



(a) (b) (c)

Fig. 1. Some training examples from (a) MNIST, (b) CalTech 101 Silhouettes and (c) Semeion datasets.

4.3 Experimental Setup

In this work, we compared the Firefly Algorithm against with the well-known PSO, Harmony Search [5] and its variant known as Improved Harmony Search (IHS) [11], which are meta-heuristic algorithms inspired in the improvisation process of music players.

In order to provide a statistical analysis by means of Wilcoxon signed-rank test [19], we conducted a cross-validation with 20 runnings. We employed 5 agents over 50 iterations for convergence considering all techniques. Table 1 presents the parameter configuration for each optimization technique[9].

Finally, we have set each DBN parameter according to the following ranges: $n \in [5, 100]$, $\eta \in [0.1, 0.9]$, $\lambda \in [0.1, 0.9]$ and $\alpha \in [0.00001, 0.01]$. Therefore, this means we have used such ranges to initialize the optimization techniques, as well as to conduct the baseline experiment by means of randomly set values. We also have employed $T = 10$ as the number of epochs for DBN learning weights procedure with mini-batches of size 20. In order to provide a more precise experimental validation, all DBNs were trained with three different algorithms[10]: Contrastive Divergence (CD) [6], Persistent Contrastive Divergence (PCD) [17] and Fast Persistent Contrastive Divergence (FPCD) [18].

[8] https://archive.ics.uci.edu/ml/datasets/Semeion+Handwritten+Digit

[9] Notice these values have been empirically setup.

[10] One sampling iteration was used for all learning algorithms.

Table 1. Parameter configuration.

Technique	Parameters
PSO	$c_1 = 1.7$, $c_2 = 1.7$, $w = 0.7$
HS	$HMCR = 0.7$, $PAR = 0.7$, $\varrho = 10$
IHS	$HMCR = 0.7$, $PAR_{MIN} = 0.1$ $PAR_{MAX} = 0.7$, $\varrho_{MIN} = 1$ $\varrho_{MAX} = 10$
FFA	$\gamma = 1.0$, $\beta_0 = 1.0$ $\alpha = 0.2$

5 Experimental Results

This section aims at presenting the experimental results concerning DBN parameter calibration. We compared four optimization methods, as well as three distinct DBN models were used: one layer (1L), two layers (2L) and three (3L) layers. Notice that the 1L approach stands for the standard RBM. Tables II, III and IV present the average MSE results for MNIST, Caltech 101 Silhouettes and Semeion Handwritten Digit datasets, respectively. The most accurate results are in bold.

	1L			2L			3L		
	CD	PCD	FPCD	CD	PCD	FPCD	CD	PCD	FPCD
PSO	0.1057	0.1058	0.1057	0.1060	0.1059	0.1058	0.1058	0.1059	0.1058
HS	0.1059	0.1325	0.1324	0.1059	0.1061	0.1057	0.1059	0.1058	0.1057
IHS	0.0903	**0.0879**	**0.0882**	0.0885	0.0886	0.0886	0.0887	0.0885	0.0886
FFA	**0.0876**	**0.0876**	**0.0882**	**0.0876**	**0.0876**	0.0886	**0.0876**	**0.0876**	0.0885

Table 2. Average MSE over the test set considering MNIST dataset.

Clearly, one can observe FFA obtained the best results for all datasets, and it has been the sole technique in Caltech 101 Silhouettes and Semeion Handwritten datasets that obtained the most accurate result. In fact, these are the best results concerning the aforementioned datasets so far, since the results with HS and IHS were the very same ones reported by Papa et al. [15]. Interestingly, except for MNIST dataset, the best results were obtained by means of a CD-driven learning and one layer only.

One can realize the larger errors for Caltech 101 Silhouettes and Semeion Handwritten Digit datasets, since they pose a greater challenge, with more complex images, as well as with a greater diversity. Curiously, the other techniques seemed to obtain better results over these datasets by using more layers, which is somehow expected, since one can use models capable of describing better the images. However, such point can not be observed through FFA, which reported the best results with one single layer. Probably, FFA could obtain even better

	1L			2L			3L		
	CD	PCD	FPCD	CD	PCD	FPCD	CD	PCD	FPCD
PSO	0.1691	0.1690	0.1689	0.1689	0.1691	0.1688	0.1692	0.1692	0.1690
HS	0.1695	0.1696	0.1691	0.1695	0.1699	0.1693	0.1694	0.1696	0.1692
IHS	0.1696	0.1695	0.1693	0.1609	0.1607	0.1612	0.1611	0.1618	0.1606
FFA	**0.1589**	0.1598	0.1616	0.1606	0.1606	0.1635	0.1606	0.1606	0.1625

Table 3. Average MSE over the test set considering Caltech 101 Silhouettes dataset.

results if one allowed more iterations for convergence using 2 and 3 layers, for instance.

	1L			2L			3L		
	CD	PCD	FPCD	CD	PCD	FPCD	CD	PCD	FPCD
PSO	0.2128	0.2128	0.2128	0.2128	0.2128	0.2128	0.2128	0.2128	0.2127
HS	0.2128	0.2128	0.2129	0.2202	0.2128	0.2128	0.2199	0.2128	0.2128
IHS	0.2131	0.2130	0.2128	0.2116	0.2114	0.2121	0.2103	0.2109	0.2119
FFA	**0.2068**	0.2078	0.2103	0.2097	0.2096	0.2125	0.2097	0.2096	0.2115

Table 4. Average MSE over the test set considering Semeion Handwritten Digit dataset.

Although HS-based techniques are not swarm-oriented, which means they do not update all particles at a single iteration, they obtained very good results. Actually, it is expected that techniques based on swarm intelligence can obtain better results, since they interchange information among all possible solutions in the search space. On the other hand, HS-based techniques create only one new solution per time, but based on all possible solutions from the current iteration.

Previous works have highlighted IHS as one of the best techniques to learn parameters in DBNs and RBMs [15, 13, 14], but we have showed new and more accurate results obtained through FFA. As a matter of fact, Pal et al. [12] showed FFA is usually better than PSO when we have non-linear and noisy functions to be optimized, which seems to be the case addressed in this work.

One shortcoming of FFA concerns with its execution time, which is similar to PSO, since both are swarm-driven, but it is slower than HS and IHS. These latter techniques do not update all agents at each iteration, but they create a single new solution instead, which means they execute the fitness function only once per iteration. Such behaviour makes them much faster than swarm-based techniques, but having a slower convergence as well.

6 Conclusions

In this paper, we dealt with the problem of calibrating DBN parameters by means of the Firefly Algorithm. The experiments were carried out over three public datasets in the context of binary image reconstruction using DBNs with 1, 2 and 3 layers. Additionally, we also considered PSO, HS and IHS for comparison purposes.

The experiments have shown FFA obtained the best results for all datasets so far, as well as FFA required much less layers than the compared optimization techniques, which means it required less computational effort, since a single layer allowed to achieve the best results. We also believe one can obtain better results with FFA using more layers, but at the price of using more iterations for convergence. In regard to future works, we intend to investigate whether more iterations will improve FFA or not, as well as to employ other meta-heuristic-based techniques to fine-tune DBNs.

Acknowledgment

The authors would like to thank FAPESP grants #2014/24491-6, #2014/16250-9 and #2015/00801-9, and CNPq grants #303182/2011-3, #470571/2013-6 and #306166/2014-3.

References

1. Ackley, D.H., Hinton, G.E., Sejnowski, T.J.: A learning algorithm for boltzmann machines. In: Waltz, D., Feldman, J. (eds.) Connectionist Models and Their Implications: Readings from Cognitive Science, pp. 285–307. Ablex Publishing Corp., Norwood, NJ, USA (1988)
2. Bengio, Y.: Learning deep architectures for AI. Foundations and Trends in Machine Learning 2(1), 1–127 (2009)
3. Bengio, Y., Courville, A., Vincent, P.: Representation learning: A review and new perspectives. IEEE Transactions on Pattern Analysis and Machine Intelligence 35(8), 1798–1828 (2013)
4. Bishop, C.: Neural networks for pattern recognition. Oxford University Press (1995)
5. Geem, Z.W.: Music-Inspired Harmony Search Algorithm: Theory and Applications. Springer Publishing Company, Incorporated, 1st edn. (2009)
6. Hinton, G.E.: Training products of experts by minimizing contrastive divergence. Neural Computation 14(8), 1771–1800 (2002)
7. Hinton, G.E.: A practical guide to training restricted boltzmann machines. In: Montavon, G., Orr, G., Müller, K.R. (eds.) Neural Networks: Tricks of the Trade, Lecture Notes in Computer Science, vol. 7700, pp. 599–619. Springer Berlin Heidelberg (2012)
8. Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural Computation 18(7), 1527–1554 (2006)

9. Huang, D.S., Gupta, P., Zhang, X., Premaratne, P.: Time series forecasting using restricted boltzmann machine. In: Emerging Intelligent Computing Technology and Applications, pp. 17–22. Communications in Computer and Information Science, Springer Berlin Heidelberg (2012)
10. Liu, K., Zhang, L., Sun, Y.: Deep boltzmann machines aided design based on genetic algorithms. In: Yarlagadda, P., Kim, Y.H. (eds.) Applied Mechanics and Materials, chap. Artificial Intelligence, Optimization Algorithms and Computational Mathematics, pp. 848–851. Scientific.Net (2014)
11. Mahdavi, M., Fesanghary, M., Damangir, E.: An improved harmony search algorithm for solving optimization problems. Applied Mathematics and Computation 188(2), 1567–1579 (2007)
12. Pal, S.K., Rai, C.S., Singh, A.P.: Comparative study of firefly algorithm and particle swarm optimization for noisy non- linear optimization problems. International Journal of Intelligent Systems and Applications 10, 50–57 (2012)
13. Papa, J.P., Rosa, G.H., Costa, K.A.P., Marana, A.N., Scheirer, W., Cox, D.D.: On the model selection of bernoulli restricted boltzmann machines through harmony search. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 1449–1450. ACM, New York, NY, USA (2015)
14. Papa, J.P., Rosa, G.H., Marana, A.N., Scheirer, W., Cox, D.D.: Model selection for discriminative restricted boltzmann machines through meta-heuristic techniques. Journal of Computational Science 9, 14–18 (2015)
15. Papa, J.P., Scheirer, W., Cox, D.D.: Fine-tuning deep belief networks using harmony search. Applied Soft Computing pp. – (2015)
16. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In: Proceedings of the Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery. pp. 847–855. KDD '13, ACM, New York, NY, USA (2013)
17. Tieleman, T.: Training restricted boltzmann machines using approximations to the likelihood gradient. In: Proceedings of the 25th International Conference on Machine Learning. pp. 1064–1071. ACM, New York, NY, USA (2008)
18. Tieleman, T., Hinton, G.E.: Using fast weights to improve persistent contrastive divergence. In: Proceedings of the 26th Annual International Conference on Machine Learning. pp. 1033–1040. ACM, New York, NY, USA (2009)
19. Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics Bulletin 1(6), 80–83 (1945)
20. Yang, X.S.: Firefly algorithm, stochastic test functions and design optimisation. International Journal Bio-Inspired Computing 2(2), 78–84 (2010)