IEC
Institute of Electronics and Computer

Journal of Artificial Intelligence and Systems, 2019, 1, 43–59
https://iecscience.org/journals/AIS
ISSN Online: 2642–2859

Soft-Tempering Deep Belief Networks Parameters Through Genetic Programming

Gustavo H. de Rosa[1], **João P. Papa**[1]

[1]Recogna Laboratory, School of Sciences, Department of Computing, São Paulo State University, Bauru, SP, Brazil
Email: {gustavo.rosa, joao.papa}@unesp.br
*Corresponding Author: Gustavo H. de Rosa, Email: gustavo.rosa@unesp.br

How to cite this paper: G. H. Rosa, J. P. Papa (2019) Soft-Tempering Deep Belief Networks Parameters Through Genetic Programming. Journal of Artificial Intelligence and Systems, 1, 43–59.
https://doi.org/10.33969/AIS.2019.11003

Received: June 12, 2019
Accepted: July 19, 2019
Published: July 24, 2019

Abstract

Deep neural networks have been widely fostered throughout the last years, primarily on account of their outstanding performance in various tasks, such as objects, images, faces, and speeches recognition. However, such complex models usually require large-scale datasets for training purposes; otherwise, they can get overfitted and therefore not achieve consistent results over unseen data. Another problem among deep models concerns their hyperparameter setting, which may require an experienced user and much effort to calibrate them, despite being application-dependent. In this paper, we present an evolutionary-inspired optimization, known as Genetic Programming, regarding Deep Belief Networks hyperparameter selection, where the terminal nodes encode the hyperparameters of the model, and proper function nodes allow an excellent combination of mathematical operators. The experimental results over distinct datasets showed Genetic Programming could outperform some state-of-the-art results obtained through other meta-heuristic techniques, thus showing to be an exciting alternative to them.

Keywords

Machine Learning, Deep Belief Networks, Optimization, Evolutionary Algorithms, Genetic Programming

1. Introduction

Machine learning techniques habitually suffer from overfitting under the lack of data, which may cause premature convergence and thus poor generalization over unseen data. The problem gets critical when dealing with deep learning architectures since such complex models often require lots of data for parameter learning. In order to overcome this issue, diverse approaches can be highlighted, such as regularization, data augmentation, and hyperparameter fine-tuning. Since this work focuses on the latter approaches, we concentrated the literature review in the works that deal with such a problem.

In the context of parametrized machine learning techniques, one shall refer to two different denominations: (i) *parameters* and (ii) *hyperparameters*. Usually, the first term stands for low-level parameters that are not controlled by the user, such as the connection weights in neural networks. Another term refers to high-level parameters that can be adjusted and chosen by the user, such as the weight decay and the hidden neurons layer's size, among

others. Both terms (i.e., parameters and hyperparameters) are of crucial importance to amend the performance of neural models. The training step of iterative-based learners can be conceived as an optimization task, where the problem is to select a feasible set of parameters that minimizes some criterion function. In some models, the hyperparameters are not taken into account in the optimization process, thus needing to be defined before the training process.

Nevertheless, the hassle of hyperparameter fine-tuning in deep learning models as a meta-heuristic-driven optimization task has received attention only recently. In 2015, Rosa et al. [1] conducted a study on how to use the Harmony Search (HS) [2] algorithm to select hyperparameters in CNNs for general-purpose datasets. Also, in that same year, Papa et al. [3] proposed an HS-based optimization to fine-tune Restricted Boltzmann Machines (RBMs) concerning binary image reconstruction, and another work using HS to optimize Discriminative RBMs in the context of binary image classification [4]. In 2016, Papa et al. [5] performed a related work regarding HS-based Deep Belief Networks (DBNs) optimization. Additionally, Rosa et al. [6] proposed the usage of a novel meta-heuristic optimization technique, the Firefly Algorithm (FA) [7] in the context of DBNs hyperparameters tuning, while Rodrigues et al. [8] proposed a Cuckoo Search-based variant regarding the same context. Furthermore, Papa et al. [9] presented a quaternion-based Harmony Search algorithm to fine-tune RBM-based models, while, in 2017, Passos et al. [10] proposed an approach to fine-tune infinity RBMs.

Regarding evolutionary-oriented optimization techniques, Yang et al. [11] used a Genetic Algorithm (GA) alongside Neural Networks for feature selection purposes. Additionally, Genetic Programming (GP) [12] was also employed in the same context, by representing distinct classifiers with different subsets of features [13, 14]. Surprisingly, there are a few works that attempted at using evolutionary-based techniques in the context of Restricted Boltzmann Machines. Liu et al. [15] applied a GA-based optimization to find the most suitable architecture (number of visible and hidden units) for Deep Boltzmann Machines (DBMs) in the context of handwritten digits, while Levy et al. [16] used GA to evolve an RBM model's weights in the context of automatic paintings classification.

An interesting approach that has not been developed yet is the usage of GP-based optimization in the context of DBN hyperparameters fine-tuning. GP uses a combination of mathematical operators to produce its outcomes, generating a broader range of values for each parameter and being extremely capable of exploring the search space and finding more suitable hyperparameters. Therefore, the main contributions of this work are twofold:

- to introduce GP in the context of DBNs hyperparameters fine-tuning;
- to fulfill the lack of research regarding evolutionary-based meta-heuristic optimization.

The remnant of this article is arranged as follows. Sections 2 and 3 describe the theoretical basis about RBMs and DBNs, and the proposed approach for Genetic Programming-inspired hyperparameter fine-tuning, respectively. Sections 4 and 5 discusses the methodology adopted in this paper and the experiments, and Section 6 states conclusions and future works.

2. Deep Belief Networks

In this section, we define the fundamental notions regarding Deep Belief Networks, as well as the theoretical background of RBMs, which are the basis for DBNs understanding.

2.1. Restricted Boltzmann Machines

Restricted Boltzmann Machines are physics-inspired neural networks. They are energy-based stochastic architectures constituted of two neuronal layers, visible and hidden, where neurons in the same layer do not share any connections. Additionally, its learning procedure is evaluated in an unsupervised way. Figure 1 illustrates an example of an RBM model,

having a visible layer $\mathbf{v}$ composed of m neurons and a hidden layer $\mathbf{h}$ with n units. Moreover, a weight matrix $W_{m\times n}$ connects the visible and hidden units, being w_{ij} the weight between visible neuron v_i and hidden neuron h_j.

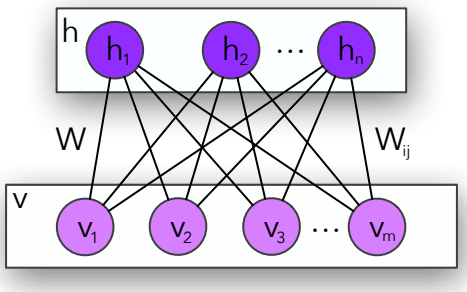


Figure 1. The RBM architecture.

Let $\mathbf{v}$ and $\mathbf{h}$ be the visible and hidden units, respectively. Additionally, let them be binary units, i.e., $\mathbf{v} \in \{0,1\}^m$ and $\mathbf{h} \in \{0,1\}^n$. One can calculate the energy function of an RBM using Equation 1, as follows:

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

where $\mathbf{a}$ and $\mathbf{b}$ are the biases of visible and hidden units, respectively. Moreover, one can use Equation 2 to calculate the probability of a joint configuration $(\mathbf{v},\mathbf{h})$:

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

where its denominator is a normalization factor that stands for all possible visible and hidden units configurations. Fundamentally, the RBM learning algorithm estimates $\mathbf{W}$, $\mathbf{a}$ and $\mathbf{b}$ values.

An optimization algorithm, such as the gradient ascent, is used to optimize the log-likelihood of the training samples and estimate proper $\mathbf{W}$, $\mathbf{a}$ and $\mathbf{b}$ values. Considering a single visible unit, it is possible to compute its probability over all possible hidden vectors using Equation 3, as follows:

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

Furthermore, Equations 4, 5 and 6 compute the weights and biases derivates in order to update them:

$$\frac{\partial \log P(\mathbf{v})}{\partial w} = P(\mathbf{h}|\mathbf{v})\mathbf{v}^T - P(\tilde{\mathbf{h}}|\tilde{\mathbf{v}})\tilde{\mathbf{v}}^T, \tag{4}$$

$$\frac{\partial \log P(\mathbf{v})}{\partial a} = \mathbf{v} - \tilde{\mathbf{v}}^T, \tag{5}$$

$$\frac{\partial \log P(\mathbf{v})}{\partial b} = P(\mathbf{h}|\mathbf{v}) - P(\tilde{\mathbf{h}}|\tilde{\mathbf{v}}), \tag{6}$$

where $\tilde{\mathbf{h}}$ and $\tilde{\mathbf{v}}$ stand for the reconstructed hidden and visible units, respectively.

In practical terms, Equation 7 calculates the probability $P(\mathbf{h}|\mathbf{v})$, which stands for the probability of obtaining $\mathbf{h}$ given the visible vector $\mathbf{v}$.

$$P(h_j = 1|\mathbf{v}) = \sigma\left(\sum_{i=1}^{m} w_{ij} v_i + b_j\right), \tag{7}$$

where $\sigma(\cdot)$ is the logistic sigmoid function. Similarly, the probability of obtaining $\mathbf{v}$ given the $\mathbf{h}$ is computed using Equation 8:

$$P(v_i = 1|\mathbf{h}) = \sigma\left(\sum_{j=1}^{n} w_{ij} h_j + a_i\right). \tag{8}$$

Therefore, it is straightforward to formulate Equation 9, which updates the weight matrix $\mathbf{W}$:

$$\mathbf{W}^{t+1} = \mathbf{W}^t + \eta(P(\mathbf{h}|\mathbf{v})\mathbf{v}^T - P(\tilde{\mathbf{h}}|\tilde{\mathbf{v}})\tilde{\mathbf{v}}^T), \tag{9}$$

where $\mathbf{W}^t$ is the weight matrix at time step t, and η is the learning rate. Moreover, Equations 10 and 11 correspond to the visible and hidden units biases update formulae, respectively:

$$\mathbf{a}^{t+1} = \mathbf{a}^t + \eta(\mathbf{v} - \tilde{\mathbf{v}}) \tag{10}$$

and

$$\mathbf{b}^{t+1} = \mathbf{b}^t + \eta(P(\mathbf{h}|\mathbf{v}) - P(\tilde{\mathbf{h}}|\tilde{\mathbf{v}})), \tag{11}$$

where $\mathbf{a}^t$ and $\mathbf{b}^t$ are the visible and hidden units biases at time step t, respectively.

Afterward, Hinton [17] proposed a weight decay hyperparameter λ responsible for penalizing weights with large magnitude[1], as well as a momentum hyperparameter α, used to control any possible falterings throughout the learning step. Therefore, Equation 9 is rewritten in Equation 12, as follows:

$$\mathbf{W}^{t+1} = \mathbf{W}^t + \underbrace{\eta(P(\mathbf{h}|\mathbf{v})\mathbf{v}^T - P(\tilde{\mathbf{h}}|\tilde{\mathbf{v}})\tilde{\mathbf{v}}^T) + \delta}_{=\Delta\mathbf{W}^t}, \tag{12}$$

where δ is computed as follows:

$$\delta = -\lambda\mathbf{W}^t + \alpha\Delta\mathbf{W}^{t-1}. \tag{13}$$

Finally, visible and hidden units biases are adjusted using Equations 14 and 15, respectively:

$$\mathbf{a}^{t+1} = \mathbf{a}^t + \underbrace{\eta(\mathbf{v} - \tilde{\mathbf{v}}) + \alpha\Delta\mathbf{a}^{t-1}}_{=\Delta\mathbf{a}^t} \tag{14}$$

and

$$\mathbf{b}^{t+1} = \mathbf{b}^t + \underbrace{\eta(P(\mathbf{h}|\mathbf{v}) - P(\tilde{\mathbf{h}}|\tilde{\mathbf{v}})) + \alpha\Delta\mathbf{b}^{t-1}}_{=\Delta\mathbf{b}^t}. \tag{15}$$

[1]During the convergence process, weights may increase and difficult the learning procedure.

2.2. Stacked Restricted Boltzmann Machines

Fundamentally, DBNs are neural network architectures constituted of a group of stack RBMs, being each layer greedily trained, i.e., an RBM at a particular layer does not acknowledge others throughout its learning process. Figure 2 describes a DBN model, being each RBM at a specific layer the one portrayed in Figure 1. Considering that, one can see a DBN as a model fashioned of L layers, being $\mathbf{W}^i$ the weight matrix of an RBM at layer i. Moreover, it is possible to perceive that the hidden units at layer i convert into the input units to the layer $i+1$. Even though it was not illustrated or mentioned in Figure 2, there are also bias units for the visible (input) and each one of the hidden layers.

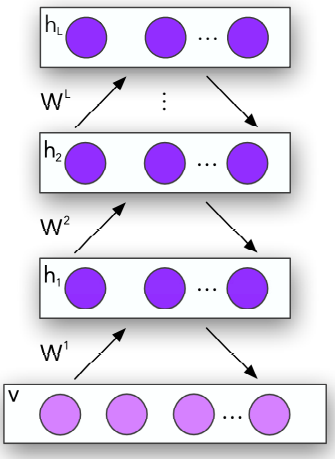


Figure 2. The standard DBN architecture.

Additionally, the learning procedure introduced by Hinton et al. [18] also employs a final fine-tuning procedure after each RBM training. Such optimization is accomplished through a Gradient descent or Backpropagation algorithm, adapting the weights matrices $\mathbf{W}^i$, where $i = 1, 2, \ldots, L$. The optimization algorithm tries to minimize a fitness function, e.g., an error measure, based on the output of an extra layer situated at the top of DBN's architecture. One can see this layer often formed of logistic or softmax units, or even a supervised classifier.

3. Genetic Programming

Genetic Programming [12] is an evolutionary optimization algorithm, which uses principles of Darwin's Theory of Evolution and biologically-inspired operators in order to create high-level solutions of particular problems. Essentially, a common challenge in computer science areas is how to solve a problem without prior knowledge from it. GP addresses this issue by automatically creating solutions that might solve the problem and genetically breeding these solutions in order to achieve more proper resolutions.

Nevertheless, there are some fundamental differences between GP and standard Genetic Algorithms (GA). A typical GP's solution is depicted by a tree comprised of terminal and function nodes, which is illustrated by Figure 3. Necessarily, the terminal nodes represent constant value, while the function nodes are the mathematical operators applied over the terminal nodes in order to assess the trees. During its evolution process, several operations are performed over the current population to produce a new set of more-adapted individuals, such as (i) selection, (ii) reproduction, (iii) mutation and (iv) crossover.

For every iteration, the operations mentioned above are employed throughout the population. Initially, the current generation best individuals are selected and reproduced,

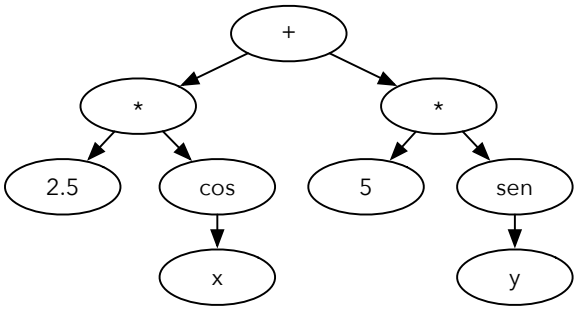


Figure 3. A GP solution representing the expression $2.5cos(x) + 5sen(y)$.

generating a possible better-tailored individual and maintaining the best ones over the generations. Afterward, mutation and crossover procedures attempt at providing a variability factor in the population. In other words, mutation changes an individual gene without considering the population, e.g., creating a new aleatory branch, while crossover switch branches within two distinguished trees. Finally, after a termination criterion is satisfied, the best individual (solution) is harvested from the tree, providing the most feasible knowledge required to solve the problem.

4. Methodology

The present approach intends to select a feasible set of DBN hyperparameters that minimizes the reconstruction error of the training set. Afterward, this set of parameters is employed to reconstruct the test samples. Essentially, the idea is to delegate the burden of empirically choosing machine learning algorithms' hyperparameters to a nature-inspired optimization technique.

The source code used in this work comes from three C language libraries: LibDEEP[2], LibOPT[3] [19], and LibDEV[4]. LibDEEP is a collection of machine learning-based techniques, containing all the implementation regarding RBMs, DBNs, and some additional techniques not used in this work. LibOPT is an assortment of meta-heuristic optimization techniques, which is designed to work with any functions' minimization. Finally, as C language is not a trivial integration language, LibDEV was developed to mitigate this issue. It is an integration library built to assimilate LibDEEP, and LibOPT together, allowing it to combine functions from both libraries and shape new algorithms, e.g., meta-heuristic optimization with DBNs.

4.1. Encoding Hyperparameters with GP

The individual (solution) vector is constituted of four decision variables to be optimized: (i) learning rate (η), (ii) number of hidden units (n), (iii) weight decay (λ) and (iv) momentum (α). These parameters are used during the DBN learning procedure, as presented in Equations 12, 14 and 15. Therefore, each terminal node encodes a four-dimensional feature vector $[\eta, n, \lambda, \alpha]$, which is then combined with others through mathematical operators to evolve into a better solution along with the iterations. Figure 4 depicts an example of a typical GP individual related to our problem, where one can observe two different terminal nodes: (i) one composed of the decision variables to be optimized, and (ii) another that contains constant variables within the range of each decision variable (fictitious values for the sake of explanation).

[2] https://github.com/jppbsi/LibDEEP
[3] https://github.com/jppbsi/LibOPT
[4] https://github.com/jppbsi/LibDEV

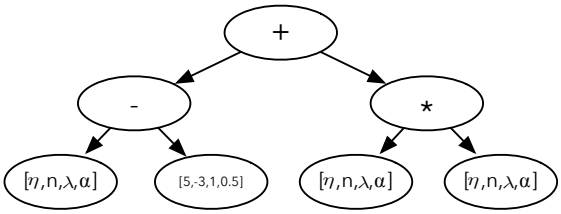


Figure 4. A GP-encoded hyperparameter tree (individual).

Regarding the fitness function, since we are dealing with the problem of binary image reconstruction, we used the well-known Mean Squared Error (MSE) to calculate the divergence between original reconstructed images. Therefore, the lower the MSE values, the better is the set of hyperparameters. In short, our optimization problem aims at finding the DBN hyperparameters that minimize the MSE over the training set, which is defined by Equation 16:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{I}_i - I_i)^2, \tag{16}$$

where $\hat{I}_i$ and I_i stand for the i^{th} reconstructed and original images, respectively.

Roughly speaking, the whole procedure can be expressed as following: given an initial set of individuals (trees) generated at random, then, we evaluate each one to obtain a final solution vector, i.e. $[\eta^*, n^*, \lambda^*, \alpha^*]$, which is then used as the current hyperparameters for the DBN. Furthermore, the model is trained and evaluated over the training set, computing an MSE value, and associating it with that particular solution vector. This process is executed for every single individual from the current population (iteration), which is then modified by reproduction, mutation, and crossover procedures to generate a new population. The algorithm executes once again until the convergence criterion is met. After that, the selected set of parameters is then applied to reconstruct the test images. Figure 5 illustrates the above procedure.

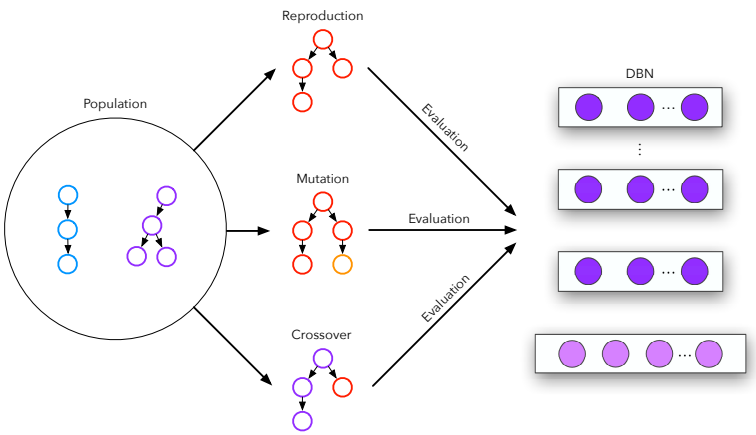


Figure 5. Proposed approach for DBN hyperparameter fine-tuning.

4.2. Datasets

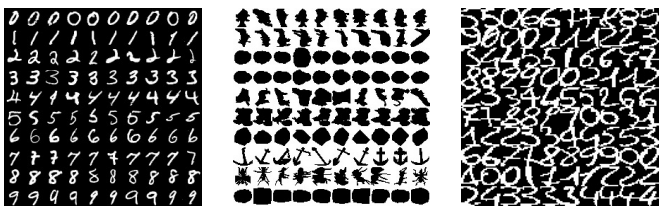
In this work, three well-known public datasets were employed, as follows:

- MNIST [20]: assembled of handwritten '0'-'9' digits images. The full version holds a training set with 60,000 images, as well as a testing set with 10,000 images. We

employed the full testing set along with a reduced training set (2%), both with a 14×14 resolution.

- CalTech 101 Silhouettes [21]: based on the former Caltech 101 dataset, comprising silhouettes of images from 101 classes with a resolution of 28×28. The training and testing sets are composed of $1,185$ and $2,307$ samples, respectively. Note that we have only used 30% of the original training set.
- Semeion Handwritten Digit [22]: composed of $1,593$ grayscale images from handwritten '0' - '9' digits written in two styles: accurate and fast ways. Furthermore, each pixel was binarized according to a 0.5 threshold and stretched into a resolution of 16×16. Note that we have used 477 images for the training set and $1,116$ images for the testing set.

Figure 6 illustrates a few training samples from the datasets mentioned above. Furthermore, no extra pre-processing has been applied to any dataset. The descriptions mentioned above stand for their original versions. As we are using the LibDEEP, LibOPT, and LibDEV environment, all datasets need to be in a binary format[5] and are available at Recogna's Laboratory website[6].



(a) (b) (c)

Figure 6. Training samples from (a) MNIST, (b) CalTech 101 Silhouettes and (c) Semeion datasets.

4.3. Experimental Setup

This article presents a comparison between Genetic Programming and nine approaches introduced by Papa et. al [5], as follows:

- Random Search (RS);
- Random Search Hyperopt (Hyper-RS) [23];
- Tree of Parzen Estimators Hyperopt (Hyper-TPE) [23];
- Harmony Search (HS) [2];
- Improved Harmony Search (IHS) [24];
- Global-Best Harmony Search (GHS) [25];
- Novel Global Harmony Search (NGHS) [26];
- Self-Adaptive Global Best Harmony Search (SGHS) [27];
- Parameter-Setting-Free Harmony Search (PSF-HS) [28].

Regarding the meta-heuristic techniques, 5 agents over 50 convergence iterations and the same parameters proposed by Papa et al. [5] were employed. Regarding GP parameters, Table 1 describes the employed configuration.

In an attempt to evaluate the sturdiness of hyperparameters fine-tuning, we assessed three unique DBN architectures: one layer, two layers, and three layers. Note that a one-layered model stands for the naïve RBM. Each DBN hyperparameter is arranged according to the following: $n \in [5, 100]$, $\eta \in [0.1, 0.9]$, $\lambda \in [0.1, 0.9]$ and $\alpha \in [0.0, 0.001]$. These intervals were used to bootstrap the optimization algorithms, as well as to perform the baseline random search experiment. Moreover, each DBN was trained during $T = 100$ epochs with

[5] https://github.com/jppbsi/LibDEEP/wiki/OPF-file-format-for-datasets
[6] http://recogna.tech

Table 1. GP parameters configuration.

Parameters	Description
Number of trees	5
Iterations	50
Tree creation method	GROW [12] with $[2,5]$ depths
Rates	Crossover $= 0.4$ | Mutation $= 0.3$ | Reproduction $= 0.3$
Functions nodes	SUM, SUB, MUL, DIV and SQRT
Terminal nodes	$1,000$ random generated numbers

mini-batches of size equal to 20. Additionally, in an effort to furnish a more accurate experimental validation, each DBN was trained with three distinct learning algorithms[7]: Contrastive Divergence (CD) [29], Persistent Contrastive Divergence (PCD) [30] and Fast Persistent Contrastive Divergence (FPCD) [31]. Finally, in order to provide statistical robustness, we conducted two-fold cross-validation with 20 runnings and a Wilcoxon signed-rank test[8] (5% significance) [32] over the experiments.

5. Experiments and Results

This section exhibits the experimental results concerning the proposed methodology.

5.1. MNIST Dataset

Table 2 describes the average MSE values regarding MNIST dataset considering DBNs with one, two, and three layers, as well as DBNs trained with CD, PCD, and FPCD learning algorithms. The bolded cells are the best results, according to Wilcoxon's signed-rank test. One can perceive that Genetic Programming achieved the foremost results for all configurations. Nevertheless, a single algorithm, IHS with PCD, has been statistically similar to GP. The best value of 0.0875 was obtained by GP using CD and FPCD for all layers' configuration.

Another way to monitor the performance of a DBN during learning is to consider the pseudo-likelihood logarithm $log(PL)$ of the training set at each iteration (epoch), i.e., $log(P(\mathbf{v}))$, which is computed over Equation 3. Figure 7 depicts the value of PL concerning different sampling techniques among the iterations during the learning procedure of a DBN with one layer, using the best set of parameters found (Figure 8). Nearly to 100 iterations, one can observe CD obtained the lowest value, which means it can generalize better over that data and is consistent with the results presented in Table 2.

In order to have a better look at the GP individuals, Figure 8 depicts one best tree obtained during the optimization procedure concerning MNIST dataset with one layer. One can observe some terminal nodes labeled with constant values, which mean the ones generated at random as aforementioned in Section 4.3. We have trees that are much smaller than this one, but we opted to show this individual in order to highlight the broad range of possibilities GP can bring us. Also, the terminal nodes labeled as $[\eta, n, \lambda, \alpha]$ store the random values used to initialize them at the beginning of the optimization process.

[7] A single sampling iteration ($k = 1$) was used for all algorithms.
[8] The test was conducted in Matlab, using the signrank() function.

Table 2. Mean reconstruction errors for the testing set regarding MNIST dataset.

	1 Layer			2 Layers			3 Layers		
Technique	**CD**	**PCD**	**FPCD**	**CD**	**PCD**	**FPCD**	**CD**	**PCD**	**FPCD**
RS [5]	0.1105	0.1101	0.1102	0.1105	0.1101	0.1096	0.1108	0.1099	0.1096
Hyper-RS [5]	0.1062	0.1062	0.1060	0.1062	0.1062	0.1060	0.1062	0.1061	0.1062
Hyper-TPE [5]	0.1059	0.1059	0.1058	0.1059	0.1059	0.1057	0.1050	0.1051	0.1051
HS [5]	0.1059	0.1325	0.1324	0.1059	0.1061	0.1057	0.1059	0.1058	0.1057
IHS [5]	0.0903	**0.0879**	**0.0882**	0.0885	0.0886	0.0886	0.0887	0.0885	0.0886
GHS [5]	0.1063	0.1062	0.1063	0.1061	0.1063	0.1061	0.1063	0.1065	0.1062
NGHS [5]	0.1066	0.1066	0.1063	0.1065	0.1062	0.1062	0.1069	0.1064	0.1062
SGHS [5]	0.1067	0.1067	0.1062	0.1072	0.1066	0.1063	0.1068	0.1065	0.1064
PSF-HS [5]	0.1005	0.1006	0.0998	0.1032	0.0976	0.1007	0.0992	0.0995	0.0998
GP	**0.0875**	**0.0875**	0.0898	**0.0875**	**0.0875**	**0.0877**	**0.0875**	**0.0875**	**0.0879**

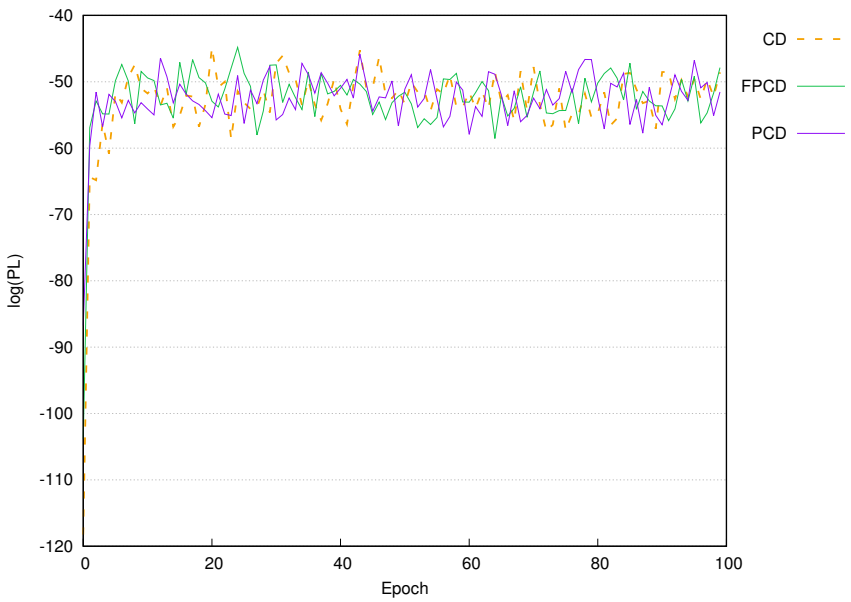


Figure 7. Epochs x Pseudo-likelihood concerning an one-layered DBN over MNIST dataset.

5.2. CalTech 101 Silhouettes Dataset

Table 3 depicts the reconstruction errors over CalTech 101 Silhouettes dataset. Once again, GP obtained the best results in all situations, achieving the lowest error so far (0.1605), and IHS obtained similar values using two and three layers, i.e., over deeper models. In this dataset, such results may lead us to conclude that deeper models optimized properly through different techniques tend to be similar. Such behavior cannot be perceived in MNIST dataset since CalTech poses a greater challenge, which can be observed by larger error values.

Additionally, Figure 9 illustrates the convergence of the log(pseudo-likelihood) concerning different training algorithms among the iterations during the learning procedure of a DBN with three layers. Note that the parameters employed to construct this graph were

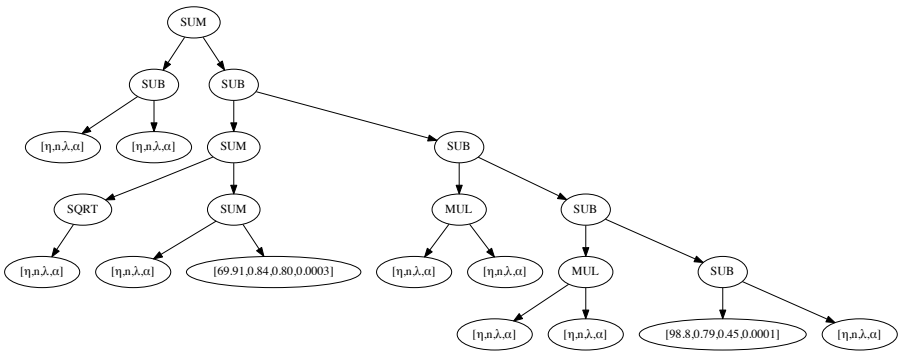


Figure 8. Best individual concerning an one-layered DBN optimization trained with CD over MNIST dataset.

Table 3. Mean reconstruction errors for the testing set regarding CalTech 101 Silhouettes dataset.

	1 Layer			2 Layers			3 Layers		
Technique	**CD**	**PCD**	**FPCD**	**CD**	**PCD**	**FPCD**	**CD**	**PCD**	**FPCD**
RS [5]	0.1755	0.1759	0.1743	0.1758	0.1755	0.1748	0.1766	0.1766	0.1742
Hyper-RS [5]	0.1696	0.1697	0.1694	0.1662	0.1662	0.1695	0.1652	0.1651	0.1650
Hyper-TPE [5]	0.1694	0.1693	0.1691	0.1693	0.1693	0.1691	0.1649	0.1642	0.1642
HS [5]	0.1695	0.1696	0.1691	0.1695	0.1699	0.1693	0.1694	0.1696	0.1692
IHS [5]	0.1696	0.1695	0.1693	0.1609	**0.1607**	**0.1612**	**0.1611**	**0.1618**	**0.1606**
GHS [5]	0.1699	0.1697	0.1692	0.1699	0.1698	0.1695	0.1697	0.1696	0.1694
NGHS [5]	0.1706	0.1703	0.1697	0.1697	0.1703	0.1694	0.1701	0.1699	0.1695
SGHS [5]	0.1703	0.1703	0.1701	0.1709	0.1706	0.1700	0.1708	0.1703	0.1701
PSF-HS [5]	0.1663	0.1670	0.1670	0.1689	0.1691	0.1681	0.1675	0.1684	0.1686
GP	**0.1605**	**0.1605**	**0.1606**	**0.1605**	**0.1605**	**0.1606**	**0.1605**	**0.1605**	**0.1607**

the ones illustrated by Figure 10. Even though PCD obtained the most significant PL value from all three algorithms, it still was capable of reconstructing the test samples at the same level than the other two.

5.3. Semeion Handwritten Digit Dataset

Table 4 exhibits the results over Semeion dataset, where GP achieved the best results so far. This dataset appears to be the most difficult one, since we can observe larger errors, thus highlighting the robustness of Genetic Programming under such situations. Figure 12 depicts one best tree concerning DBN optimization over Semeion dataset with two layers. Now, one can observe the constant-valued terminal nodes contain eight values each, instead of the four presented in Figure 8, since we have four values to optimize per layer (Section 4.1).

Moreover, one can attain to Figure 11 and perceive that PCD obtained the lowest PL value within all three learning algorithms. Furthermore, regardless of the training algorithm, all techniques were capable of achieving statistically similar reconstruction errors, showing their suitability on solving the intended task.

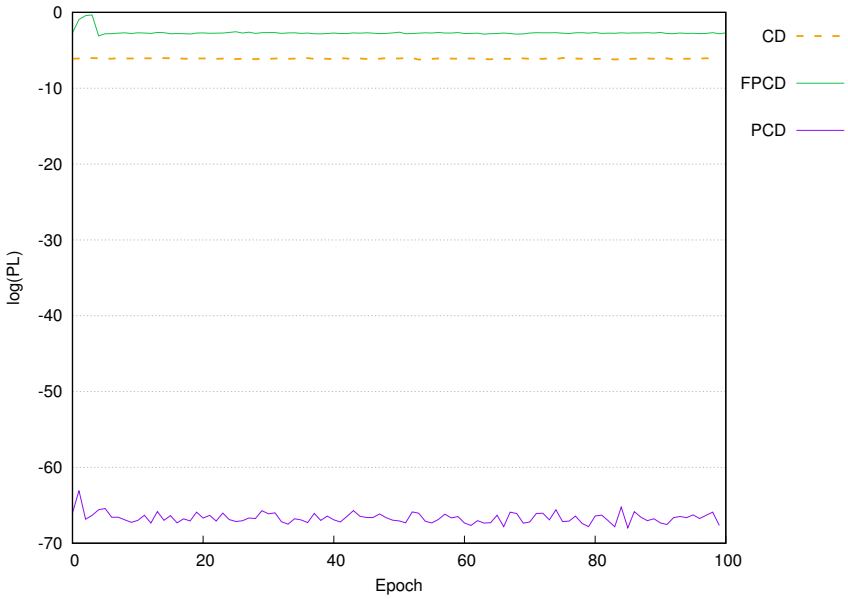


Figure 9. Epochs x Pseudo-likelihood concerning a three-layered DBN over CalTech dataset.

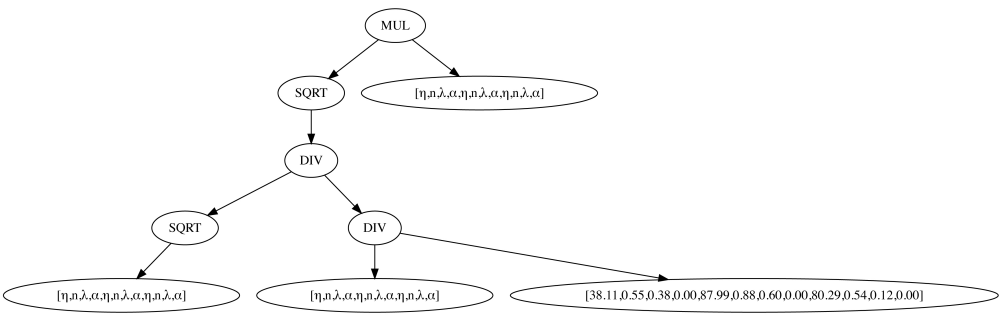


Figure 10. Best individual concerning a three-layered DBN optimization trained with PCD over CalTech dataset.

5.4. Discussion

The experiments over the already-mentioned datasets allow us to draw some important conclusions. Firstly, one can use any optimization technique instead of empirically setting DBN hyperparameters by hand. Nevertheless, this comes with a higher computational load, being only useful when there is no prior knowledge about the problem. Secondly, GP seemed to be the most effective technique among all comparisons, showing promise information about evolutionary-based algorithms in the context addressed in this paper.

Additionally, as Papa et al. [5] do not describe the time spent in the experiments, one can refer to Table 5 in order to assess how many calls each optimization technique made to the fitness function[9]. Even though GP made the most amount of call to the fitness function (almost five times more), it achieved the lowest reconstruction errors, being a feasible optimization algorithm to be employed in DBNs fine-tuning.

[9]One can understand a single call to the fitness function as a whole DBN training procedure.

Table 4. Mean reconstruction errors for the testing set regarding Semeion dataset.

	1 Layer			2 Layers			3 Layers		
Technique	**CD**	**PCD**	**FPCD**	**CD**	**PCD**	**FPCD**	**CD**	**PCD**	**FPCD**
RS [5]	0.2146	0.2143	0.2145	0.2146	0.2144	0.2139	0.2143	0.2140	0.2140
Hyper-RS [5]	0.2127	0.2129	0.2129	0.2129	0.2129	0.2129	0.2129	0.2129	0.2128
Hyper-TPE [5]	0.2128	0.2128	0.2128	0.2128	0.2128	0.2127	0.2128	0.2128	0.2128
HS [5]	0.2128	0.2128	0.2129	0.2202	0.2128	0.2128	0.2199	0.2128	0.2128
IHS [5]	0.2131	0.2130	0.2128	0.2116	0.2114	0.2121	0.2103	0.2109	0.2119
GHS [5]	0.2133	0.2129	0.2128	0.2129	0.2130	0.2129	0.2129	0.2129	0.2128
NGHS [5]	0.2134	0.2132	0.2131	0.2130	0.2131	0.2129	0.2131	0.2132	0.2130
SGHS [5]	0.2135	0.2131	0.2130	0.2131	0.2131	0.2130	0.2132	0.2132	0.2130
PSF-HS [5]	0.2137	0.2130	0.2130	0.2121	0.2120	0.2124	0.2120	0.2120	0.2121
GP	**0.2084**	**0.2090**	**0.2095**	**0.2096**	**0.2095**	**0.2097**	**0.2095**	**0.2095**	**0.2096**

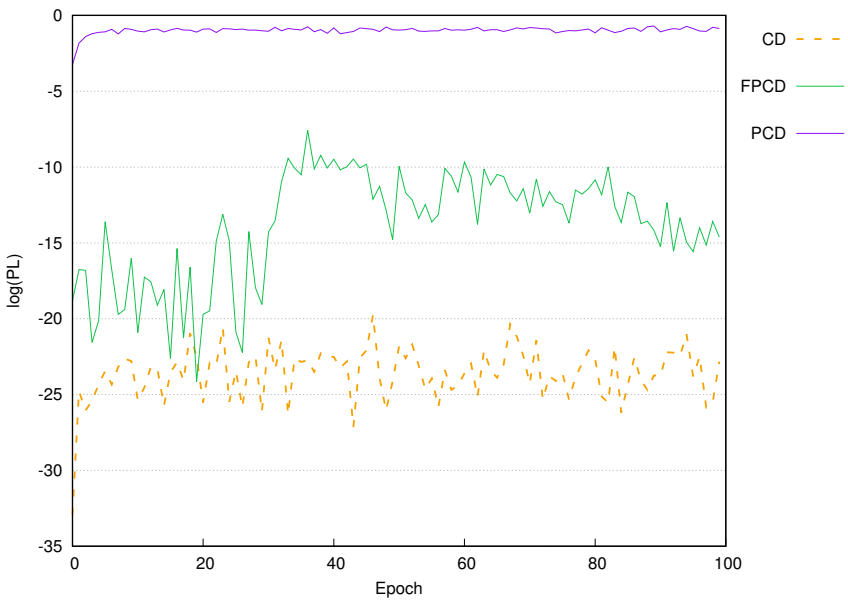


Figure 11. Epochs x Pseudo-likelihood concerning a two-layered DBN over Semeion dataset.

6. Conclusions

Deep Belief Networks have been widely employed for numerous purposes throughout the last years due to their capacity in representing unknown data by encoding different sources of information. Nevertheless, there are only a few works that aim at solving the problem of model selection for such techniques, i.e., how to learn the most feasible set of parameters that led to the best results. Usually, they are accomplished by empirical evaluations, random values, or even hand-choosing, which may be time-consuming and not always the best choices. In this article, we proposed to assess the bottleneck of DBN hyperparameters fine-tuning through Genetic Programming. The intent is to fashion the issue of selecting suitable Deep Belief Networks hyperparameters concerning binary images reconstruction as an evolutionary-based optimization task and compare against nine other optimization

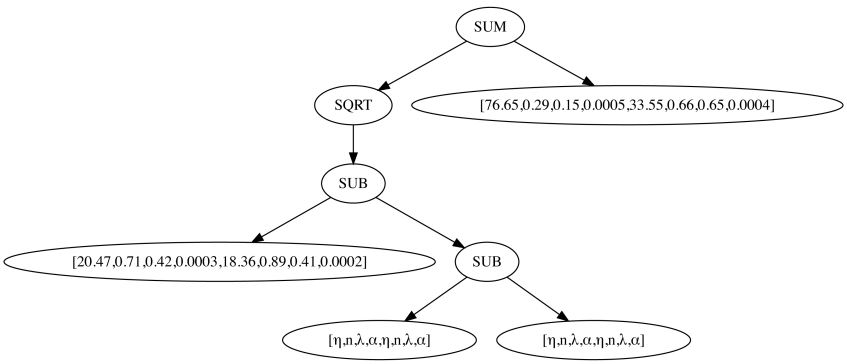


Figure 12. Best individual concerning a two-layered DBN optimization trained with FPCD over Semeion dataset.

Table 5. Amount of calls to the fitness function for each optimization algorithm.

Algorithm	Number of calls
RS	1
Hyper-RS	55
Hyper-TPE	55
HS	55
IHS	55
GHS	55
NGHS	55
SGHS	55
PSF-HS	55
GP	255

approaches proposed by Papa et al. [5].

In order to achieve the goal of this work, we carried out experiments using three public datasets in the context of binary image reconstruction. Although all datasets are shape-oriented, one can perceive that Semeion Handwritten Digit is the most challenging one, due to its complex shapes and higher reconstruction errors. The experiments were validated throughout a 20-runnings cross-validation process, and with three different learning algorithms: Contrastive Divergence, Persistent Contrastive Divergence, and Fast Persistent Contrastive Divergence. Regarding MNIST dataset, GP was able to obtain the best results over almost all configurations, being only comparable with IHS at PCD and FPCD in a one-layered DBN, while for the CalTech 101 dataset, GP was also able to achieve the best results, being statistically similar to IHS in five occasions, i.e., two-layer DBN (PCD and FPCD) and three-layer DBN (all learning algorithms). Considering the Semeion Handwritten Digit dataset, GP was solely able to achieve the best results, emphasizing the plausibility of using evolutionary-based optimization techniques in order to properly select machine learning algorithms hyperparameters, instead of empirically choosing these values.

Finally, we think we could accomplish the principal purpose of this work, which is

to foster evolutionary-based algorithms to fine-tune DBN hyperparameters. Although GP provided a higher computational load, it was capable of achieving the lowest reconstruction errors among all comparisons, being suitable to be used when there is no knowledge about the problem. Regarding future works, we aim at evaluating other variants of GP-based techniques to the context addressed in this paper. Additionally, we also plan to apply GP to Discriminative Deep Belief Networks, evaluating its performance in the context of binary image classification.

Acknowledgements

The authors appreciate FAPESP grants #2013/07375-0, #2014/12236-1, #2016/19403-6, #2017/02286-0, #2017/25908-6, #2018/21934-5 and #2019/02205-5, and CNPq grants 307066/2017-7 and 427968/2018-6.

Conflicts of Interest

The authors acknowledge that there is no conflict of interest concerning the disclosure of this article.

References

[1] G. H. Rosa, J. P. Papa, A. N. Marana, W. Scheirer, and D. D. Cox, "Fine-tuning convolutional neural networks using harmony search," in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, ser. Lecture Notes in Computer Science, 2015, vol. 9423, pp. 683–690, 20th Iberoamerican Congress on Pattern Recognition.

[2] Z. W. Geem, *Music-Inspired Harmony Search Algorithm: Theory and Applications*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[3] J. P. Papa, G. H. Rosa, K. A. P. Costa, A. N. Marana, W. Scheirer, and D. D. Cox, "On the model selection of bernoulli restricted boltzmann machines through harmony search," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '15. New York, USA: ACM, 2015, pp. 1449–1450.

[4] J. P. Papa, G. H. Rosa, A. N. Marana, W. Scheirer, and D. D. Cox, "Model selection for discriminative restricted boltzmann machines through meta-heuristic techniques," *Journal of Computational Science*, vol. 9, pp. 14–18, 2015.

[5] J. P. Papa, W. Scheirer, and D. D. Cox, "Fine-tuning deep belief networks using harmony search," *Applied Soft Computing*, vol. 46, pp. 875–885, 2016.

[6] G. H. Rosa, J. P. Papa, K. A. P. Costa, L. A. Passos, C. R. Pereira, and X. S. Yang, "Learning parameters in deep belief networks through firefly algorithm," in *Artificial Neural Networks in Pattern Recognition: 7th IAPR TC3 Workshop, ANNPR*, 2016, pp. 138–149.

[7] X. S. Yang, "Firefly algorithm, stochastic test functions and design optimisation," *International Journal Bio-Inspired Computing*, vol. 2, no. 2, pp. 78–84, 2010.

[8] D. Rodrigues, X. S. Yang, and J. P. Papa, "Fine-tuning deep belief networks using cuckoo search," in *Bio-Inspired Computation and Applications in Image Processing*, X. S. Yang and J. P. Papa, Eds. Academic Press, 2016, pp. 47–59.

[9] J. P. Papa, D. R. Pereira, A. B., and X. S. Yang, *On the Harmony Search Using Quaternions*. Cham: Springer International Publishing, 2016, pp. 126–137.

[10] L. A. Passos and J. P. Papa, "Fine-tuning infinity restricted boltzmann machines," in *2017 30th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, Oct 2017, pp. 63–70.

[11] J. Yang and V. Honavar, "Feature subset selection using a genetic algorithm," in *Feature extraction, construction and selection*. Springer, 1998, pp. 117–136.

[12] J. Koza, *Genetic programming: on the programming of computers by means of natural selection*. Cambridge, USA: The MIT Press, 1992.

[13] J. Y. Lin, H. R. Ke, B. C. Chien, and W. P. Yang, "Classifier design with feature selection and feature extraction using layered genetic programming," *Expert Systems with Applications*, vol. 34, no. 2, pp. 1384–1393, 2008.

[14] R. Ramirez and M. Puiggros, "A genetic programming approach to feature selection and classification of instantaneous cognitive states," in *Workshops on Applications of Evolutionary Computation*. Springer, 2007, pp. 311–319.

[15] K. Liu, L. M. Zhang, and Y. W. Sun, "Deep boltzmann machines aided design based on genetic algorithms," in *Applied Mechanics and Materials*, vol. 568. Trans Tech Publ, 2014, pp. 848–851.

[16] E. Levy, O. E. David, and N. S. Netanyahu, "Genetic algorithms and deep learning for automatic painter classification," in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*. ACM, 2014, pp. 1143–1150.

[17] G. E. Hinton, "A practical guide to training restricted boltzmann machines," in *Neural Networks: Tricks of the Trade*, ser. Lecture Notes in Computer Science, G. Montavon, G. Orr, and K. R. Müller, Eds. Springer Berlin Heidelberg, 2012, vol. 7700, pp. 599–619.

[18] G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[19] J. P. Papa, G. H. Rosa, D. Rodrigues, and X. S. Yang, "Libopt: An open-source platform for fast prototyping soft optimization techniques," *arXiv preprint arXiv:1704.05174*, 2017.

[20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[21] B. Marlin, K. Swersky, B. Chen, and N. Freitas, "Inductive principles for restricted boltzmann machine learning," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 509–516.

[22] "Semeion handwritten digit data set," Semeion Research Center of Sciences of Communication, via Sersale 117, 00128 Rome, Italy, Tattile Via Gaetano Donizetti 1-3-5, 25030 Mairano (Brescia), Italy, Tech. Rep., 2008.

[23] J. S. Bergstra, D. Yamins, and D. D. Cox, "Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms," in *Python for Scientific Computing Conference*, 2013, pp. 1–7.

[24] M. Mahdavi, M. Fesanghary, and E. Damangir, "An improved harmony search algorithm for solving optimization problems," *Applied Mathematics and Computation*, vol. 188, no. 2, pp. 1567–1579, 2007.

[25] M. G. Omran and M. Mahdavi, "Global-best harmony search," *Applied Mathematics and Computation*, vol. 198, no. 2, pp. 643–656, 2008.

[26] D. Zou, L. Gao, J. Wu, S. Li, and Y. Li, "A novel global harmony search algorithm for reliability problems," *Computers & Industrial Engineering*, vol. 58, no. 2, pp. 307–316, 2010, scheduling in Healthcare and Industrial Systems.

[27] Q. K. Pan, P. Suganthan, M. F. Tasgetiren, and J. Liang, "A self-adaptive global best harmony search algorithm for continuous optimization problems," *Applied Mathematics and Computation*, vol. 216, no. 3, pp. 830–848, 2010.

[28] Z. W. Geem and K. B. Sim, "Parameter-setting-free harmony search algorithm," *Applied Mathematics and Computation*, vol. 217, no. 8, pp. 3881–3889, 2010.

[29] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Computation*, vol. 14, no. 8, pp. 1771–1800, 2002.

[30] T. Tieleman, "Training restricted boltzmann machines using approximations to the likelihood gradient," in *Proceedings of the 25th International Conference on Machine Learning*. New York, NY, USA: ACM, 2008, pp. 1064–1071.

[31] T. Tieleman and G. E. Hinton, "Using fast weights to improve persistent contrastive divergence," in *Proceedings of the 26th Annual International Conference on Machine Learning*. New York, NY, USA: ACM, 2009, pp. 1033–1040.

[32] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.