ISSN 1870-4069

Evolutionary Training of Deep Belief Networks for Handwritten Digit Recognition

Susana Espinoza-Pérez[1], Alfonso Rojas-Domínguez[1], S. Ivvan Valdez-Peña[2], Luis Ernesto Mancilla-Espinoza[1]

[1] Tecnológico Nacional de México - Instituto Tecnológico de León, Mexico

[2] Universidad de Guanajuato, Depto. de Ingeniería Electrónica, Mexico

alfonso.rojas@gmail.com

Abstract. Two of the most representative deep architectures are Deep Convolutional Neural Networks and Deep Belief Networks (DBNs). Both of these can be applied to the problem of pattern classification. Nevertheless, they differ in the training method: while the first is trained by backpropagation of the error through the whole network, the latter is typically pre-trained on a per-layer basis using an unsupervised algorithm known as Contrastive Divergence (CD), and then it is fine-tuned with a gradient descent algorithm. Although metaheuristic algorithms have been widely applied for hyperparameter tuning, little has been published regarding alternative methods to pre-train DBNs. In this work, we substitute the conventional pre-training method with an evolutionary optimization algorithm called the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES). The pretraining is achieved by minimizing the so-called reconstruction error. This proposal is validated on the problem of MNIST digit recognition by training a Deep Belief Network, following the methodology described by Hinton and Salakhutdinov (Science, 2006). It is also compared against the well known Genetic Algorithm (GA). We provide evidence to show that, although the computational cost is significantly highern than CD, the use of CMA-ES leads to a significantly smaller reconstruction error than CD and the GA.

Keywords. Deep learning, deep belief networks, deep autoencoders, metaheuristics, evolutionary algorithms.

1 Introduction

Pattern recognition can be defined as the classification of the data based on the acquired knowledge or statistical information extracted from patterns and its representations [15]. Traditional classifiation methods, such as K-means, Support Vector Machine (SVM), Artificial Neural Networks (ANNs) and linear methods, require the knowledge, contained in a dataset, to be provided to them as features. Therefore, a fundamental step in pattern classification is the selection (and extraction) of features or attributes that best represent the dataset, known as patterns. Typically, an expert does this selection [2]. However, in recent years these experts have been substituted by Deep Neural Networks (DNNs) that are capable of computing and selecting those features during their training process [10].

Despite the fact that deep learning techniques are effectively used as feature learners, some issues arise regarding their training, for instance, since they usually have complex architectures, they are more susceptible to overfitting than simpler models. Depending on the model, they can easily contain hundreds of thousands of parameters that must be calibrated or fine-tuned [22][23]. Fine-tuning the parameters of a machine learning model can be posed as an optimization task, in which the fitness function is the effectiveness of the technique over some validating set, e. g., the best parameters in a DNN are those that show the best classification result for a given problem.

Since about 2006, researchers have been attracted by Restricted Boltzmann Machines (RBMs) due to their simplicity, high level of parallelism and strong representation abilities [22]. RBMs can be interpreted as stochastic neural networks, being mainly used for image reconstruction, in which their hidden layer learns a probability distribution over an input dataset presented to the vsible layer [16]. Few years after their initial proposal, Hinton et al. realized that one can obtain more complex representations by stacking together a few RBMs, thus leading to the so-called Deep Belief Networks (DBNs) [22]. Recently, several works have been published that focuse on tuning hyperparameters of DBNs based on nature-inspired metaheuristics, such as Harmony Search [17], Cuckoo Search [21], Artificial Bee Colony Algorithm [9], and Firefly Algorithm [22]. Last year, Papa et. al. [23] employed Quaternion-based Harmony Search to tune hyperparameters of DBNs and evaluated some metaheuristic techniques to tune the so called Infinity Discriminative Restricted Boltzmann Machines in the context of binary images [18].

Besides nature-inspired metaheuristics, there are other types of metaheuristics, such as Evolutionary Algorithms (EAs). The EAs can be classified into different families, for instance: Evolution Strategies (ESs) [5], Genetic Algorithms [12] and Estimation of Distribution Algorithms (EDAs) [24]. ESs were introduced by Rechenberg (1973) and further developed by Schwefel (1981); they are applied to problems in continuous domains [13]. EDAs were introduced in the field of evolutionary computation for the first time by MŰhlenbein and PaaB (1996). All of these are population-based techniques. In general, EDAs possess the advantage that they do not need extra parameters to perform the optimization; instead, a new population is sampled from a probability distribution, which is estimated from the problem database [13]. The *Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES)* [5], uses a multivariate normal distribution to search for the optimum. CMA-ES is an algorithm which, actually, shares characteristics of EAs and EDAs.

Despite the satisfactory results that have been published of metaheuristics used for the optimization of the hyperparameters of DBNs, no studies have been found where the same metaheuristics are utilized for the optimization of the weights of these networks. In this work, the use of the CMA-ES algorithm is proposed for such a task, in the context of training RBMs in an unsupervised manner, which is known as pre-training these to be used in a latter phase of a DBNs designed for classification. The technique is tested on the problem of handwritten digit classification and compared against the typical training procedure of RBMs.

The rest of this paper is organized as follows. Section 2 briefly presents all the techniques that are involved in this work. Section 3 describes our proposal and methodology. Section 4 reports our experimental results and finally, Section 5 offers our conclusions and directions for future work.

2 Materials and Methods

This section introduces all of the required technical background to understand the experimental comparison presented in this work. First the Boltzmann machines and Deep Belief networks are introduced, then the training algorithm known as Contrastive Divergence is briefly explained, and finally, the metaheuristics compared in this work are reviewed.

2.1 Boltzmann Machines and Deep Belief Nets

Boltzmann Machines are symmetrical, undirected bipartite graphical models [11] containing neuron-like (typically binary) units that perform simple computations with which they are capable of learning internal representations. Their name comes from their sampling distribution being the Boltzmann distribution, popular in statistical mechanics for modelling energy states of a material.

The Boltzmann or Gibbs distribution is shown in Eq. (1). Notice that the maximum (probability) of the Bolztmann is the minimum of $f(x)$. Additionally, the probability mass of x depends on the T value , if T is infinite, the Boltzmann becomes a uniform distribution, on the other hand, if T goes to 0, the Boltzmann becomes a Diract delta:

$$p(x) = \frac{\exp(-f(x)/kT)}{Z}. \tag{1}$$

These properties have been exploited by researchers in computer science, by means of using the Boltzmann distribution for estimating the *minimum* of an energy function $f(x)$. Because such minimum is the most probable state, it would be sampled with a high probability if an adequate T value is chosen. Nevertheless, a direct sampling from the Bolztmann distribution requires dealing with the whole range of $f(x)$, which in turn is an exhaustive search. Fortunately, there are indirect sampling methods whose samples converge to the Boltzmann distribution, one of them is the Gibbs sampler, that takes advantage of neighborhood structures of a graphical model to iteratively generate samples that asymptotically become distributed in agreement with the Boltzmann distribution. If an arbitrary T value is chosen, the sampler could be biased towards a random

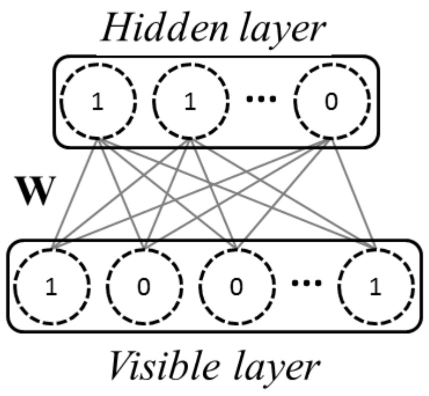

Fig. 1. A Restricted Boltzmann Machine. The matrix $\mathbf{W}$ represents the weights or connections between the computational binary units of different layers. These connections are undirected.

suboptimal point. To avoid this problem, often a slowly decreasing sequence of values for $\{T\}$ is used. This technique is called a *simulated annealing.*

Because of this, the Boltzmann machines are also classified as "energy based models" and can be seen as the stochastic, generative counterpart of Hopfield networks. In theory, a Boltzmann machine could model the distribution of data presented to it after running for long enough time at a certain energy level. In practice, however, the training time required for the machine to learn grows exponentially with the number of units in the network, and learning becomes impractical. This disadvantage is overcome using an architecture called a "Restricted Boltzmann Machine" (RBM), where the computational units are organized in layers called a visible (input) layer and a hidden layer, and intralayer connections between the units forming the network are prohibited. Thus, in an RBM there are only dependencies between units of different types (see Fig. 1).

Training of one RBM containing two layers (one visible, one hidden) is carried out via a simple algorithm generally known as Contrastive Divergence [6], which will be explained in detail in Section 2.1. Several RBMs can be stacked together one after the other in a daisy-chain fashion, where the hidden layer of one RBM becomes the visible layer of the next RBM. This strategy makes it possible to efficiently train what are called Deep Autoencoders containing many layers for dimensionality reduction and also, by adding a further supervised-learning fine tuning training step, Deep Belief Networks (DBN), which can be used for pattern recognition (e.g. in digital images and video). Each new layer added to a DBN improves the overall generative model much like other models such as multilayer perceptrons (MLPs) or convolutional neural networks (CNNs). Training of a DBN can thus be studied by looking at the training of the stacked RBMs contained within. The propagation function z_i of hidden unit i is:

$$z_i = \sum_j w_{ij} v_j + b_i, \tag{2}$$

where w_{ij} is the symmetric weight on the connection between unit i and unit j in the visible layer, v_j can take the values 0 (the visible unit is OFF) or 1 (the visible unit is ON), and b_i represents a bias added to each of the hidden units. Based on z_i, the hidden unit i will turn on with a probability given by the logistic function in Eq. (3). Although not very common, other types of variables can be used:

$$p(v_i = 1) = \frac{1}{1 + e^{-z_i}}. \tag{3}$$

The same pair of relationships, Eqs. 2 and 3 govern the behaviour of visible units with respect to the hidden units, thus representing the probability that a binary latent variable has a value of 1 during generation (top-down, from hidden to visible units) or inference (bottom-up, from visible to hidden units). If the units in alternating layers are thus sequentially updated, the network will eventually reach a stationary distribution, which is a Boltzmann distribution described in Eq. (4); notice that this is the same distribution as in Eq. (1) for $x = s$, $E(s) = f(x)/T$, and $Z = \sum_q e^{-E(\mathbf{q})}$, which is the normalization constant. In that case, the probability of a state vector $\mathbf{s}$, can be determined solely by the energy of that state vector, normalized by the sum of the energies of all possible binary state vectors $\mathbf{q}$, called the partition function:

$$P(\mathbf{s}) = \frac{e^{-E(s)}}{\sum_q e^{-E(\mathbf{q})}}. \tag{4}$$

A lower energy corresponds to a more desirable configuration of the network, so that training the network means minimizing its energy, defined in Eq. (5):

$$E(\mathbf{s}) = -\sum_i v_i(\mathbf{s}) b_i - \sum_{i<j} w_{ij} v_i(\mathbf{s}) v_j(\mathbf{s}), \tag{5}$$

where $v_i(\mathbf{s})$ represents the binary state assigned to unit i by $\mathbf{s}$. In the next subsection, the necessary steps to perform the training of an RBM by updating its weights are described.

2.2 The Contrastive Divergence Algorithm

During training of a RBM[1], its weights w_{ij} are adjusted via Eq. (6):

$$\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} + \varDelta\mathbf{W} = \mathbf{W}^{(t)} + \eta \frac{\partial log(p(v))}{\partial w_{ij}}, \tag{6}$$

where $\mathbf{W}$ represents the matrix of weights w_{ij}, $\varDelta\mathbf{W}$ is the adjustment for $\mathbf{W}$ at iteration $(t+1)$ and η is known as the learning rate. This updating rule is an approximation to the maximum likelihood method, where the gradient is given by:

$$\frac{\partial log(p(v))}{\partial w_{ij}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}, \tag{7}$$

where v_i represents a *visible* node, h_j represents a *hidden* node and the angled brackets represent the expected value with respect to either the distribution of the data or the model. While computing the distribution with respect to the data is straightforward, the expectation with respect to the model requires the use of extended alternating Gibbs sampling between the visible and the hidden units. The training algorithm developed by Hinton, and known as "Contrastive Divergence" (CD) uses only n steps of Gibbs sampling ($n = 1$ works sufficiently well) [6] . The pseudo-code for CD is shown in Algorithm 1.

Algorithm 1 Pseudo-code of the Contrastive Divergence algorithm

```
1: Set the values of the training units to a training vector (binary data is requiered).
2: Update the hidden units given the visible units: $p(h_i = 1|\mathbf{V})$ according to Eq. (3).
3: Update the visible units given the hidden units: $p(v_j = 1|\mathbf{H})$ according to Eq. (3).
4: Re-update the hidden units given the newly obtained values for the vissible units,
   (redo step 2 using the new $\mathbf{V}$).
5: Perform weight adjustment $\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} + \Delta\mathbf{W}$ with Eq. (6).
6: return $\mathbf{W}$.
```

In order to demonstrate the benefits of performing pre-training on the feature extraction module in a DBN, the classification performance of the network with and without the pretraining stage is contrasted in Fig. 2. It can be observed that from the very beginning of the fine-tuning stage, the pre-trained network rapidly achieves a significantly lower error than the network without pre-training. After epoch 10, the networks improve their performance very slowly or practically have converged to their final performance. Applying the fine-tuning for many more epochs does not produce any further improvement.

As can be appreciated, CD is a very simple method for unsupervised training of an RBM. Nevertheless, it is an iterative method that requires a lot of data. In this work we test the hypothesis of whether population-based metaheuristics can perform better than CD. Furthermore, we intend to generate indications about the most adequate metaheuristic to perform the training process of an RBM. In this vein, stochastic algorithms which use descent directions are known as well-performing methods in the context of convolutional neural networks [14]. Based on such evidence, we propose to use the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES), which is a population based metaheuristic that employs a probability distribution where the maximum-variance is oriented towards the direction of steepest descent. In order to test our hypothesis and our proposal, the performance of CD is contrasted against that of the CMA-ES and against the Genetic Algorithm (GA), which is arguably the most widely

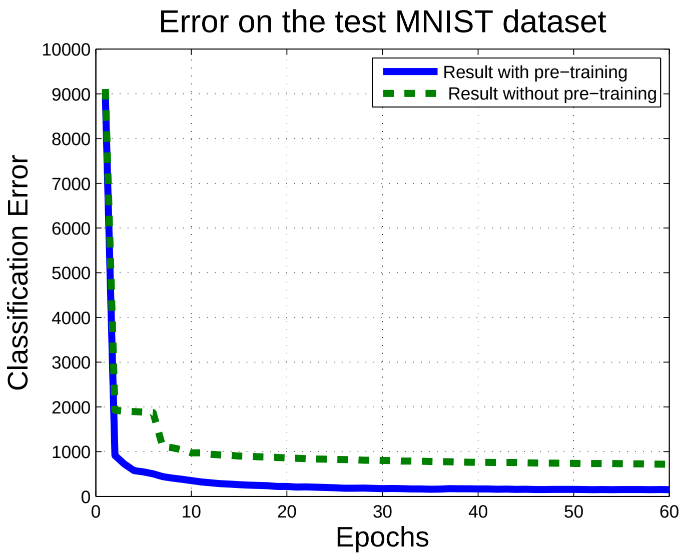

Fig. 2. Effect of employing pre-training against random weight-initialization in the classification performance.

used metaheuristic and, consequently, a suitable basis for comparison. These evolutionary algorithms are described in the following subsection.

2.3 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are based on the principles of biological evolution. They evolve a population of candidate solutions (individuals), by applying reproduction and variation operators on a selected set (parent set), this set is selected with a bias to the best solutions. The intention is to produce fitter offspring for the next generation. The offspring replace the current population, usually, preserving the best solution through generations, this is called elitism. The process is repeated until a stopping criterion is met. The EAs can be classified into different families, for instance: Evolution Strategies (ES) [5], Genetic Algorithms [12] and Estimation of Distribution Algorithms (EDAs) [24], all of them are evolutionary algorithms but each of them has a particular way of working.

The *Genetic Algorithm (GA)* is arguably the most popular of the Evolutionary Algorithms. Initially developed by Holland [8] and popularized through the work of Goldberg [4], GAs are now widely applied in science and engineering as adaptive algorithms for solving practical problems. The GA is also a population-based technique inspired by natural selection and genetics. This technique maintains a population of individuals called chromosomes and formed by "genes" in some D-dimensional search space. Adaptation strategies (selection, crossover and mutation) are used to make the individuals evolve in search for the global optimum. In this way a new population is created and the process is repeated in a new generation. In GA, the selection operator chooses a pair of chromosomes for crossover, combining some of the genes of each of the parents. Only the best of the new chromosomes is passed on to the next generation (elitism). The mutation alters a chromosome, creating a new one. The process above is summarised in Algorithm 2.

Algorithm 2 Genetic Algorithm Pseudo-code

```
 1: Initialize chromosome population.
 2: Set $P_r \leftarrow$ Probability of reproduction.
 3: Set $P_m \leftarrow$ Probability of mutation.
 4: while stopCrit $\neq$ true do
 5:     Evaluate the population.
 6:     Choose two parent chromosomes.
 7:     if $P_r$ then
 8:         Apply the crossover operator.
 9:     end if
10:     if $P_m$ then
11:         Apply the mutation operator.
12:     end if
13:     Accept the new solution if its fitness is lower.
14: end while
15: return $\mathbf{W}_l$
```

The *Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES)* [5] uses a multivariate normal distribution to search for the optimum, it is an algorithm which, actually, shares characteristics of EAs and EDAs. In continuous domains, ESs [3], usually, produce the offspring set by generating Normal deviates with mean at a parent position, and a covariance matrix often adapted in two fashions: 1) by using evolutionary operators on the parameters which determine the size and rotation of the covariance matrix, 2) by adapting the covariance matrix using improvement directions. The first is named as the self-adapted-parameter fashion while the second is the covariance matrix adaptation (CMA).

In the CMA, a set of parent solutions is used to determine the size, position and orientation of a normal distribution used to sample the candidate solutions. One of its most important characteristics is that the orientation of the multivariate normal model aligns the search to promising regions, in a kind of descent path.

The CMA-ES pseudo-code is shown in Algorithm 3, steps 1-3 initialize the multivariate Gaussian distribution and the algorithm parameters. In steps 5-7, the new candidate solutions are generated from the multivariate Gaussian distribution and evaluated on the objective function f. In step 9, the population is sorted selecting the best solutions. The previous mean is stored in step 10, then, the new mean, the isotropic path p_σ, and the anisotropic path p_c, are computed in steps 11-13. The isotropic and anisotropic path include information about the historical direction of the search, which has been computed using the best solutions, the first is a kind of covariance-normalized direction while the second is an absolute direction. Lastly, in steps 14-15 the new covariance matrix is computed, and the isotropic and anisotropic paths are used to modify it,in order to guide the search toward the improvement directions, hence, it is expected that the maximum variance direction has a large projection over the maximum improvement direction. Steps 5-15 compose the main iteration of the CMA-ES, these steps are repeated until a stopping criterion is met.

Algorithm 3 Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) Pseudocode

```
 1: Set $\lambda \leftarrow$number of samples per iteration, at least two, generally $> 4$
 2: Initialize $m, \sigma, C = I, p_\sigma = 0, p_c = 0$
 3: $t \leftarrow 0$
 4: while stopCrit $\neq$ true do
 5:     for $i = 1 : \lambda$ do
 6:         $x_i \leftarrow \mathcal{N}(m, \sigma^2 C)$
 7:         $\mathcal{F}_i \leftarrow f(x_i)$
 8:     end for
 9:     $x_{1\ldots\lambda} \leftarrow x_{s(1)\ldots s(\lambda)}$ with $s(i) = argsort(f(1,\ldots,\lambda), i)$
10:     $m' = m$
11:     $m \leftarrow update\_m(x_1, \ldots, x_\lambda)$
12:     $p_\sigma \leftarrow update\_ps(p_\sigma, \sigma^{-1}C^{-1/2}(m - m'), ||p_\sigma||)$
13:     $p_c \leftarrow update\_pc(p_c, \sigma^{-1}(m - m'), ||p_\sigma||)$
14:     $C \leftarrow update\_C(C, p_c, (x_1 - m')/\sigma, \ldots, (x_\lambda - m')/\sigma)$
15:     $\sigma \leftarrow update\_sigma(\sigma, \ldots, ||p_\sigma||)$
16: end while
17: return $x_1$
```

3 Methodology

In this work, we substitute the traditional method of performing the pre-training, with evolutionary optimization algorithms, the CMA-ES and the well known GA, for a direct comparison of their abilities to perform the pre-training of three RBMs stacked together to form a DBN. The network pre-training is achieved by minimizing the reconstruction error, Eq. (8). This proposal is validated on the problem of MNIST digit recognition by training a Deep Belief Network, following the methodology described by Hinton and Salakhutdinov [7] and is also compared against the well known Genetic Algorithm (GA). All of these techniques were described in Section 2.

The MNIST (Modified National Institute of Standards and Technology) database is a moderately large database of handwritten digits, that has become very popular in the field of machine learning and particularly, deep learning. The database contains instances of digits, each in the form of an image of size

Fig. 3. A sample of the MNIST handwritten digits.

28×28 pixels. There are 60,000 training images and 10,000 testing images in the database (Fig. 3).

The problem of classification of MNIST digits consists of building a system that automatically provides the true class of the images in the testing set. This can be accomplished using a set of RBMs stacked one after the other, as explained in Section 2.1, followed by a "completely connected" or "fully connected" (FC) layer with ten neurons (one per digit class) that provide the predictions of the system [20, 7, 19]. The set of RBMs are considered the feature extraction module and are pre-trained using CD, an unsupervised method, and then the whole network (i.e. including the FC layer) is fine-tuned using the supervised method of gradient descent known as backpropagation. In this work we follow the methodology described by Hinton and Salakhutdinov [7], and modify their code, provided as Supporting Online Material where the authors employed the conjugate gradient method as an algorithm for the fine-tuning of the network.

Hinton and Salakhutdinov defined the architecture of their network with five layers of feature extraction of sizes: $(28 \times 28 = 784) \times 500 \times 500 \times 2000 \times 10$ (last is the FC layer), this is illustrated in Fig. 4-a. The authors also used a training procedure based on what is known as mini-batches, where the training dataset is partitioned into relatively small subsets (the mini-batches) that are presented to the system between weight updates in order to accelerate the training (a more typical approach would be to perform an update of the network weights after each individual instance). For reproducibility purposes, and given that they obtained a competitive result corresponding to 1.2% test error, in this work the architecture and hyper-parameters defined by Hinton and Salkhutdinov are used, with one exception described below.

A population-based metaheuristic maintains a set of candidate solutions, known as "individuals" for short, from which better solutions (or at least not-worse than the current best) are produced. Each of the candidate solutions is represented by a vector typically defined in the solution space; i.e., each

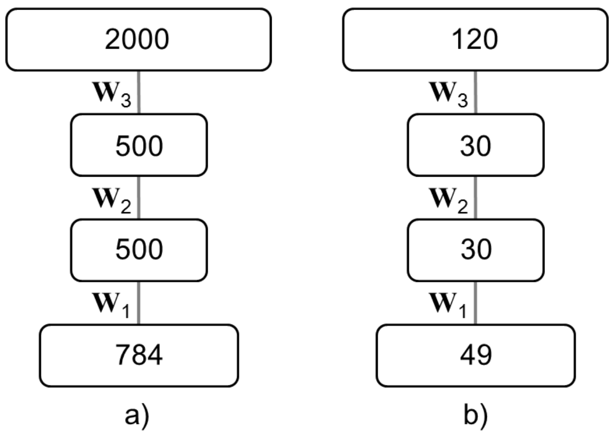

Fig. 4. (a) Architecture used in Hinton and Salakhutdinov. (b) Architecture used in this work.

individual is represented by a vector with as many elements as the dimensionality of the solution space, D. In this work, the solutions that is looked for is a series of weight matrices and bias vectors, one matrix at a time. Each of these matrices contains several thousand elements, for instance, weight matrix $\mathbf{W}_1$ in Fig. 4-a contains $784 \times 500 = 392,000$ elements. This is to say that in order to obtain the optimal $\mathbf{W}_1$, a population of individuals is to be maintained, each individual of size 392,000 elements. The CMA-ES algorithm described in Section 2.3 involves the computation and diagonalization of a covariance matrix of size $D \times D$, and this becomes prohibitively large (the data does not fit in memory) for $D = 392,000$. Although some alternatives (such as hashing algorithms [25]) exist for large-scale and high-dimensional tasks, in this work we use a simpler approach in order to provide a straightforward proof-of-concept of our contribution that can work with other similar problems in smaller dimensions: resizing the training data. By resizing the images in the MNIST dataset to one fourth of their original size, and consequently the dimensions of the network to those illustrated in Fig. 4-b, we found that it is possible for the CMA-ES algorithm to handle the problem directly. Thus, the final architecture used in the experiments is of size: $49 \times 30 \times 30 \times 120 \times 10$. The pseudocode of our experimental methodology is provided in Algorithm 4.

4 Experimental Results

In this section we report the results of performing the pre-training of the feature extraction module for the classification of MNIST digits, comparing the typical training of RBMs against using metaheuristics, in this case the CMA-ES algorithm and the GA. The main two elements for metaheuristics to function are the

Algorithm 4 Methodology for training a DBN

```
 1: Set $J \leftarrow$ input layer size,$K \leftarrow$ hidden layer size.
 2: For CD: Set $\alpha \leftarrow$ momentum, $\eta \leftarrow$ learning rate.
 3: For CMA-ES: Set $M \leftarrow$ population size.
 4: For GA: Set $M \leftarrow$ population size.
 5: Split dataset, $X$, into mini-batches, $x_i$.
 6: while stopCrit $\neq$ true do
 7:    for $i = 1 : N$ do
 8:       if CD then
 9:          Compute $E_i$ with $\mathbf{W}_l^{(t)}$.
10:          Compute $\mathit{\Delta}\mathbf{W}$, Eq. (7).
11:          Update $\mathbf{W}_l^{(t+1)} = \mathbf{W}_l^{(t)} + \mathit{\Delta}\mathbf{W}$, Eq. (6).
12:       else
13:          if CMA-ES then
14:             Create population $\mathbf{P}^{(t)} = \{\mathbf{W}_l^{(0)}, \mathbf{W}_l^{(1)}, \ldots, \mathbf{W}_l^{(M)}\}$.
15:             Compute $E_i$ with $\mathbf{P}^{(t)}$.
16:             Update population with Algorithm 3.
17:          end if
18:       else
19:          if GA then
20:             Create population $\mathbf{P}^{(t)} = \{\mathbf{W}_l^{(0)}, \mathbf{W}_l^{(1)}, \ldots, \mathbf{W}_l^{(M)}\}$.
21:             Compute $E_i$ with $\mathbf{P}^{(t)}$.
22:             Update population with Algorithm 2.
23:          end if
24:       end if
25:    end for
26:    Compute epoch error: $E = \sum_i E_i$.
27: end while
28: return $\mathbf{W}_l$
```

shape of an individual candidate solution and a fitness function to be optimized (typically, minimized). In this work, the candidate solutions are defined as a concatenation of the weight matrix of the i-th RBM, W_i (flattened), the visible layer biases, b_i, and the hidden layer biases, b_i. All of these are defined as real values. The fitness function used is the reconstruction error, which is the same function guiding the CD algorithm. To compute the reconstruction error of one RBM, input data is given to the RBM, the activation of the hidden layer is computed using Eqs. (2) and (3). Then the reconstruction, $\tilde{v}$ is computed using the same equations, but taking the hidden layer as the input and the visible layer as the output. Thus, the reconstruction error is defined as:

$$E = \sum_j (v_j - \tilde{v}_j)^2. \tag{8}$$

For each of the RBMs, the CD algorithm, the CMA-ES and the GA were executed for 50 iterations, so that the total amount of iterations allocated to train the feature extraction module is 150. The reconstruction error for each iteration was recorded. Since the iterations coincide with the presentation of all the instances of the training data, they are referred to as "epochs".

The control parameters used in the experiments for CD, CMA-ES and GA are reported in Table 1. Notice that three values are reported for the number of visible units in the visible layer, N_{vis}, and the number of units in the hidden layer, N_{hid}. The three values correspond to the three RBMs that compose our system and that are trained sequentially one after another.

Table 1. Control parameters used for the CD, CMA-ES and GA.

Contrastive Divergence	
Parameter	Value
N_{vis}	$\{49, 30, 30\}$
N_{hid}	$\{30, 30, 120\}$
Initial Population, W	$\mathcal{N}(0, 0.1)$
Initial Population, biases	zeros
Learning rate	$\eta = 0.1$
Initial Momentum, α_I	0.5
Momentum α	0.9
CMA-ES	
Parameter	Value
Problem dimension, D	$N_{vis} \times N_{hid} + N_{vis} + N_{hid}$
Initialization method	$\mathcal{N}(0, 0.5)$
Population size, λ	$4 + \lfloor 3 \log(D) \rfloor$
Recombination points	$\lambda/2$
Genetic Algorithm	
Parameter	Value
Problem dimension, D	$N_{vis} \times N_{hid} + N_{vis} + N_{hid}$
Initial Population	$\mathcal{N}(0, \sigma_l)$ for each layer l
Standard Deviation	$\sigma_1 = 0.5, \sigma_2 = \sigma_3 = 0.1$
Population size, λ	$\gamma_l(4 + \lfloor 3 \log(D) \rfloor)$, $\gamma_1 = 15, \gamma_2 = 3, \gamma_3 = 3$

The experiment was repeated 30 times and the average result is shown in Fig. 5. The plot shows the reconstruction error obtained by CD (continuous line), CMA-ES (long-dashed line) and the GA (short-dashed line). Vertical bars along the curves are used to show the standard deviation over the 30 trials.

It can be observed that CD exhibits a much smaller variability than CMA-ES. On the other hand, CMA-ES achieves significantly smaller error for the first two RBMs trained. The GA exhibits a much larger variability than CD and comparable to that of CMA-ES, except for the final portion of the epochs, where the population of CMA-ES converges but the one of GA does not. In the end (epoch 150) the mean performance of GA is within a standard deviation of the mean performance of CMA-ES, but in general CMA-ES achieves the lowest error significantly faster than GA; at epochs 85-90, CMA-ES has reached the lowest error recorded. In particular, for the first RBM, GA requires many more epochs

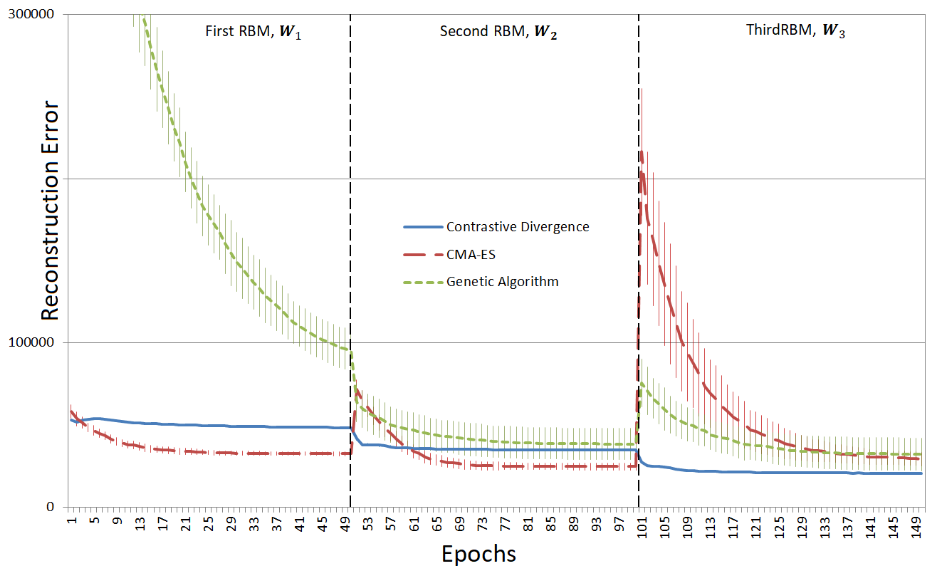

Fig. 5. Error plots comparing CD against CMA-ES and GA.

to significantly reduce the error level, so much so that the beginning of the curve lies outside of the figure. In fact, after 50 epochs one can observe that the curve of the Genetic Algorithm still points significantly downwards, indicating that the GA still has potential to improve the weight optimization beyond the 50 epochs.

Regarding other aspects of the algorithms, it must be noted that in this work the same scheme of training via mini-batches was used for both training strategies. According to this scheme, a training epoch is completed when the network is presented with all of the training data, one mini-batch at a time. While this approach is most beneficial for the Contrastive Divergence algorithm, it may not be the best way to use the training data to guide the CMA-ES and GA algorithms. For instance, one can think of employing a randomized sample of the training data to be used in the computation of an estimate of the reconstruction error, and use that estimate to guide the evolution in these algorithms. This would eliminate a considerably part of the computational cost incurred in having to evaluate all the individuals in the CMA-ES / GA population on all of the training data just to guide the exploration of the solution space. Instead of adjusting the training scheme to benefit each of the algorithms, in this work we employed the same training scheme for both algorithms for the sake of a fair comparison.

5 Conclusion and Future Work

In this work, the proposal of using metaheuristics to perform the pre-training of the feature extraction module in a DBN was put forward and tested using the CMA-ES, which is a population-based metaheuristic. DBNs were pre-trained for the problem of handwritten digit recognition using the MNIST dataset, and their performance was measured by the reconstruction error of the RBMs included in the feature extraction portion of the DBN. The performance of the CMA-ES was compared against that of CD and also against that of the GA. Our results show that CMA-ES performs the best among these algorithms and validate our hypothesis regarding the suitability of CMA-ES to successfully approximate the optimal weights of the DBN. Nevertheless, we have observed that the success of the algorithm strongly depends on the problem dimension. This becomes evident in Fig. 5 where CMA-ES outperforms the other two methods up to the beginning of the third RBM, where it completely loses its advantage and struggles to come back in track. The reason behind this is that the third RBM contains the most units and therefore substantially more weights between its layers. This has a particularly detrimental effect on the CMA-ES algorithm.

It should be noted that in this work our interest was to analyse the performance of metaheuristic techniques for the pre-training of the different RBMs in a DBN designed for handwritten digit recognition but not to observe the end result. This is because the set of stacked RBMs should undergo a second training stage after the pre-training (known as fine-tuning), to improve the classification performance. This second training process is very efficient and may obscure the existing differences between the pre-training achieved by each of the compared techniques. In a future work we will study in detail the effect of the pre-training on the convergence of the fine-tuning step.

As a general conclusion, we have determined that metaheuristics (in particular the CMA-ES), are competitive methods for pre-training a DBN using the reconstruction error. However, metaheuristics are quite demanding in terms of the computational resources (such as memory, operations and objective function evaluations) required to perform the optimization. This drawback are inherit to both metaheuristics tested, but not in the same scale. The CMA-ES requires memory of the order $O(n^2)$ for storing a covariance matrix, in consequence, the size (dimensionality) of the problems that can be tackled by this algorithm are bounded by this computational resource. This disadvantage forced us to apply a scaling to the handwritten digit images in order to reduce their dimensionality. Another disadvantage of employing CMA-ES (and any other population-based algorithm) is the cost of evaluating all the candidate solutions in the population.

In this regard, we can readily identify two main issues to investigate in future work: first, to identify other metaheuristics with a similar or better performance than CMA-ES and with lesser complexity and computational requirements. Second, to employ a different training scheme, more suitable to the particular requirements of the selected metaheuristics (instead of being tailored for CD). In any case, it can be concluded that the combination of metaheuristics and deep learning methods is a promissory line of research.

Acknowledgements. This work is supported by CONACYT, through Scholarship 482124 (S. Espinoza) and Research Grant Cátedras-2598 (A. Rojas). Alfonso Rojas is a CONACYT Research Fellow.

References

1. Ackley, D.H., Hinton, G.E., Sejnowski, T.J.: A learning algorithm for boltzmann machines. Cognitive science 9(1), 147–169 (1985)
2. Andrew R. Webb, K.D.C.: Statistical pattern reconigtion. Wiley, 3 edn. (2011)
3. Beyer, H.G., Schwefel, H.P.: Evolution strategies–a comprehensive introduction. Natural computing 1(1), 3–52 (2002)
4. Goldberg, D.E.: Genetic algorithms in search, optimization, and machine learning. Addison-Wesley (1989)
5. Hansen, N., Niederberger, A., Guzzella, L., Koumoutsakos, P.: A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. IEEE Transactions on Evolutionary Computation 13(1), 180–197 (2009)
6. Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural computation 18(7), 1527–1554 (2006)
7. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. science 313(5786), 504–507 (2006)
8. Holland, J.H.: Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press (1992)
9. Horng, M.H.: Fine-tuning parameters of deep belief networks using artificial bee colony algorithm. 2nd International Conference on Artificial Intelligence: Techniques and Applications (AITA 2017) (2017)
10. Huy Nguyen, M.L.N.: A deep neural architecture for sentence-level sentiment classification in twitter social networking. In International Conference of the Pacific Association for Computational Linguistics pp. 15–27 (2017)
11. Jordan, M.I.: Learning in Graphical Models. MIT press, Cambridge Mass (1998)
12. Klanac, A., Jelovica, J.: A concept of omni-optimization for ship structural design. Advancements in Marine Structures, Proceedings of MARSTRUCT pp. 473–481 (2007)
13. Larrañaga, Pedro, Lozano, A, J.: Estimation of distribution algorithms: A new tool for evolutionary computation, vol. 2. Springer Science & Business Media (2001)
14. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature 521(7553), 436 (2015)
15. M. Narasimha Murty, V.S.D.: Pattern Recognition, An Algorithmic Approach. Springer-Verlag London, 1 edn. (1988)
16. Papa, J.P., Rosa, G.H., Marana, A.N., David Cox, W.S.: Model selection for discriminative restricted boltzmann machines through meta-heuristic technique. Journal of Computational Science 9, 14–18 (2015)
17. Papa, J.P., Scheirer, W., Cox, D.: Fine-tuning deep belief networks using harmony search. Applied Soft Computing 46, 875–885 (2016)
18. Passos, L.A., Papa, J.P.: Fine-tuning infinity restricted boltzmann machines. In Graphics, Patterns and Images (SIBGRAPI) pp. 63–70 (2017)
19. Ranzato, M., Huang, F., Boureau, Y., LeCun, Y.: Unsupervised learning of invariant feature hierarchies with applications to object recognition. Proc. of Computer Vision and Pattern Recognition Conference (CVPR 2007) (2007)

20. Revow, M., Williams, C.K., Hinton, G.E.: Using generative models for handwritten digit recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence 18(6), 592–606 (1996)
21. Rodrigues, D., Yang, X.S., Papa, J.P.: Fine-tuning deep belief networks using cuckoo search. Bio-Inspired Computation and Applications in Image Processing pp. 47–59 (2017)
22. de Rosa, G.H., Yang, X.S., Papa, J.P., da Costa, K.A.P.: Learning parameters in deep belief networks through firefly algorithm. In IAPR Workshop on Artificial Neural Networks in Pattern Recognition pp. 138–149 (2016)
23. de Rosa, G.H., Yang, X.S., Papa, J.P., Pereira, D.R.: Quaternion-based deep belief networks fine-tuning. Applied Soft Computing 60, 328–335 (2017)
24. Valdez, S., Hernández, A., Botello, S.: A Boltzmann based estimation of distribution algorithm. Information Sciences 236, 126–137 (2013)
25. Wang, J., Liu, W., Kumar, S., Chang, S.: Learning to hash for indexing big data—a survey. Proceedings of the IEEE 104(1), 34–57 (Jan 2016)