Envisioning the Benefits of Back-Drive in Evolutionary Algorithms

Unai Garciarena
Intelligent Systems Group
University of the Basque Country, UPV/EHU
Donostia-San Sebastian, Spain
unai.garciarena@ehu.eus

Alexander Mendiburu
Intelligent Systems Group
University of the Basque Country, UPV/EHU
Donostia-San Sebastian, Spain
alexander.mendiburu@ehu.eus

Roberto Santana
Intelligent Systems Group
University of the Basque Country, UPV/EHU
Donostia-San Sebastian, Spain
roberto.santana@ehu.eus

***Abstract*—Among the characteristics of traditional evolutionary algorithms governed by models, memory volatility is one of the most frequent. This is commonly due to the limitations of the models used to guide this kind of algorithms, which are generally very efficient when sampling, but tend to struggle when facing large amounts of data to represent. Neural networks are one type of model which conveniently thrives when facing vast amounts of data, and does not see its performance particularly worsened by large dimensionality. Several successful neural generative models, which could perfectly fit as a model for driving an evolutionary process are available in the literature. Whereas the behavior of these generative models in evolutionary algorithms has already been widely tested, other neural models -those intended for supervised learning- have not enjoyed that much attention from the research community. In this paper, we take one step forward in this direction, exploring the capacities and particularities of back-drive, a method that enables a neural model intended for regression to be used as a solution sampling model. In this context, by performing extensive research into the most influential aspects of the algorithm, we study the conditions which favor the performance of the back-drive algorithm as the sole guiding factor in an evolutionary approach.**

***Index Terms*—Deep learning, Model-based evolutionary algorithms, Back-drive**

I. Introduction

Research on model-based evolutionary algorithms (MEA) [1]–[3] such as estimation of distribution algorithms (EDA) [4]–[6] has mainly focused on the use of probabilistic graphical models [7] (PGM) to represent the relationships between the variables of the problem. PGMs are a concise and usually easy to interpret representation of the dependencies between the variables of a problem. The employment of this kind of models as part of an evolutionary framework consists of two phases. Firstly, the model induction part [8]. Secondly, sampling the models, for which several efficient methods exist, particularly those based on a partial ordering of the variables for sampling [9].

This work has received support from the TIN2016-78365-R (Spanish Ministry of Economy, Industry and Competitiveness http://www.mineco.gob.es/portal/site/mineco) and Elkartek IT-1244-19 (Basque Government) programs. Unai Garciarena holds a predoctoral grant (ref. PIF16/238) from the University of the Basque Country. We gratefully acknowledge the support of NVIDIA Corporation with the donation of a Titan X Pascal GPU used to accelerate the process of training the models used in this work.

Despite their suitability and efficiency, PGMs are difficult to combine with gradient-based optimization algorithms, which can be extremely efficient for problems of many variables. Recently, the use of artificial neural network (NN) models has been increasingly investigated as a way to capture and exploit the dependencies of a problem [10]–[12]. NN-based evolutionary algorithms (EA) learn a neural model from the best solutions and use this model to generate new promising solutions. Model learning can be framed as classification, regression, or distribution learning tasks, and can exploit fast gradient-based optimization procedures for model learning.

A crucial step when using deep NNs (DNN) in model-based EAs is the way in which the generation of new solutions is performed. The implementation of this step is not straightforward since, in contrast to PGMs, NNs are difficult to interpret. This makes the task of identifying the manner in which the model maps the input to the desired output a very comlpex one. Therefore, the usage of these models for solution generation results as non-trivial. A strategy to obtain inputs designed to produce a desired prediction by a given NN is defined in [13]: NN inversion. This strategy was later adopted in [11], where it was used as a controller of an EA.

In this paper we propose the usage NNs as the probabilistic model of the MEA. Moreover, the viability of the NN inversion or back-drive algorithm for effectively sampling solutions is analyzed. Furthermore, we define different ways of training the NN model, incorporate information from previous generations, perform the back-drive, and exploit the trained model, testing these variations against each other. This results in an in-depth experimentation for determining the conditions under which back-drive offers optimal performance when used in an evolutionary framework.

The rest of this paper is organized as follows. Section II provides an overview of the DNN model and the back-drive algorithm used in this work. Section III introduces MEAs in detail. Related work is discussed in Section IV. In Section V, different aspects of the back-drive technique that need to be taken into account are mentioned. Section VI describes the experiments conducted regarding the characteristics introduced in the previous section. Section VII makes a series of insights on the aforementioned experiments. Finally, conclusions drawn from the whole process are summarized in Section VIII.

II. Background

A. Multi Layer Perceptron

The multi-layer perceptron (MLP) is the primitive NN-based model used to perform deep learning (DL) [14]. Its most basic configuration consists of three different types of layers, which are composed of neurons. The first layer, which takes the input data, the last layer, which provides a prediction regarding the information placed in the input, and the hidden layers, which are able to learn high-abstraction features of the data. In an MLP, these layers are both sequential and dense, in the sense that every neuron in layer $l-1$ is connected to every other neuron in layer l, and there exist no more connections between neurons. These layer-wise connections can be represented as matrix operations of the following form:

$$n_l = \sigma_l(w_l \times n_{l-1} + b_l) \tag{1}$$

where, n_l is a layer of neurons; the representation of the data computed in the l-th layer. $w_l, b_l \in \theta$ are the parameters learned for layer l. For the input layer, $n_{l-1} = n_0 = x$, the input data, and for the last layer, $n_l = \hat{y}$, the prediction. σ is a function, for which the usual choice is to be non-linear.

Commonly, w_l and b_l are trained employing a gradient descent method [15]. These methods differentiate an error measurement between the outcome of the net $\hat{y}$ and the desired output y, applying changes to the net parameters layer by layer, starting from the last layer and moving backwards until the first one, in an algorithm called back-propagation [16].

B. Back-drive algorithm

The network inversion [13] or back-drive [11] algorithm exploits the propagation of error through layers in a different way to back-propagation. This method is used for, given a trained model, making convenient alterations to the input data so that it matches a target prediction by the model. Applying this method allows the usage of models from a very extensive research field; supervised learning via DNN (e.g., image classification), for data generation purposes.

To contrast back-propagation and back-drive methods, in the training phase, the input data x and target value y remain constant. θ is trained so that $\hat{y}$ is as close to y as possible. The back-drive algorithm assumes an already trained θ, which is frozen from this point on. A data point x is placed in the first layer, and the error between the prediction of the model for that data point and the target prediction y^* is propagated until the data point, which is now modifiable. To put it in a simple way, instead of asking "What θ do we need for reducing $\hat{y} - y$?", we ask "Given θ, what x do we need to reduce $\hat{y} - y^*$?".

This way, data points x with arbitrary $\hat{y}$ predictions can be modified so that the new x' meet certain desired requirements.

III. Model Based Evolutionary Algorithms

Basic EAs, e.g., genetic algorithms (GA) [17], usually produce acceptable results when applied to simple problems, but tend to struggle as the complexity of the problem increases [18]. The lack of treatment of variable dependency can be one of the causes of this behavior. Designing effective recombination or sampling operators can be the solution. With this in mind, MEAs were developed as more advanced algorithms, including probabilistic models able to represent the (in) dependencies between the variables. Procedures for learning and sampling these models are required.

Algorithm 1 shows the general pseudocode of a basic MEA. The description of the functions used by the algorithm follow:

- generate_population(): Generates a population of randomly initialized individuals.
- evaluate_population(pop): Given a population, this function returns a list of the corresponding fitness values of the individuals.
- select_solutions(pop, fit): Given a population and its corresponding fitness values, this function returns the individuals from which the model is to be learned.
- create_model(pop, fitness): Given a set of solutions and their corresponding fitness values, this function returns a model, which is trained using the given information.
- sample_model(model): Given the model, this function generates samples from it.
- combine_pop(pop, fit, offs, off_fit): Given a population, new offspring, and its corresponding fitness values, this function returns a new population and fitness values.

Algorithm 1: Pseudo-code for a basic MEA.

```
pop = generate_population();
fit = evaluate_population(pop);
while halting condition is not met do
    sel_pop, sel_fit = select_solutions(pop, fit);
    model = create_model(sel_pop, sel_fit);
    offs = sample_model(model);
    offs_fit = evaluate_population(offs);
    pop, fit = combine_pop(pop, fit, offs, offs_fit)
end
```

IV. Related Work

A. NN-based EAs

With the same goal in mind (optimization) but using a different approach compared to ours, an increasing number of recent works [10], [11], [19], [20] propose the application of neural models, which sometimes involve deep architectures. Generally, as in the case of this work, the NN models are used to generate new solutions for the EA.

A fundamental difference of NNs over graphical models is that the information about the problem structure is usually represented in latent variables or distributed structures that make the interpretability of the model a difficult task.

The DNNs that have been tested for EDAs exhibit a variety of behaviors: autoencoders used in a traditional EDA scheme in [21] are extremely fast compared to methods that learn Bayesian networks, but they fail to achieve the same efficiency as BOA [22] in terms of function evaluations. When used as a mutation distribution in [19], GA-dA (see Table I) outperforms

BOA in some problems (notably on the knapsack problem) but it is outperformed on the hierarchical HIFF function.

The convenience of using DNNs is another important question to discern, given the impressive results of DNNs in other domains. One of the conclusions obtained from the evaluation of the deep Boltzmann machine (DBM) neural network in EDAs is that the effort for learning the multi-layered DBM model does not seem to pay off for the optimization process [23]. Also in [11], where DNNs with 5 and 10 layers are used as neural models, it is acknowledged that the learning process can be time consuming. While the Deep-Opt-GA is evaluated across a set of diverse artificial and real-world problems, it is not possible to determine the gain of the algorithm over EDAs since it is compared to a fast local optimizer and a GA.

A categorization of NNs relevant for the work presented in this paper is based on the strategy used to sample new solutions. Table I lists some of the main NN-based EAs, indicating whether the models used to generate solutions were conceived for generative purposes or not.

TABLE I
DESCRIPTION OF SOME OF THE MAIN NEURAL NETWORK MODELS PROPOSED FOR MODEL-BASED EAS.

Algorithm	year	NN model	Ref.	Generative	Deep
BEA	2000	HM	[24]	yes	no
MONEDA	2008	GNG	[25]	no	no
RBM-EDA	2010	RBM	[26]	no	no
REDA	2013	RBM	[27]	yes	no
DAGA	2014	DA	[28]	no	no
DBM-EDA	2015	DBM	[23]	yes	yes
AED-EDA	2015	DA	[21]	no	no
GAN-EDA	2015	GAN	[10]	yes	no
GA-dA	2016	DA	[19]	no	no
GA-NADE	2016	NADE	[19]	yes	no
RBM-EDA	2017	RBM	[20]	yes	no
Deep-Opt-GA	2017	DNN	[29]	yes	yes

B. *Sampling solutions from a regression neural network*

Using ANN inversion as a method to sample a trained network in an EA was firstly performed in [11]. In this work, the author proposed a common MLP to guide the search in an EA. At each generation, all the solutions were evaluated, and their fitness functions were scaled to [0,1]. The MLP was trained with this data, then back-driven [13] to obtain new solutions. This inversion was performed by requiring the net to modify a mutated solution so that it would return the maximum score known at each moment. After the sampling, the solution was improved by a hill climbing algorithm. A new population was generated by repeating this sample-improving method. This whole process was iterated within an evolutionary framework.

Two variations of this approach were tested against three versions of hill climbing. The proposed algorithm obtained significantly better results optimizing a simple noisy function with continuous variables, as well as for three combinatorial optimization problems. However, it was unable to obtain this kind of superiority when tested in 2-D layout problems.

Unlike in the paper discussed above - [11]-, in this paper, the back-drive algorithm is not assisted by any local-search algorithm or external mutation operator of any kind. Additionally, we perform an in-depth analysis of the behavior of the method depending on several settings applicable to the algorithm. This results in a set of guidelines to be taken into account at the time of performing further research on this topic.

V. DEEP NEURAL NETWORK AS A MODEL

One of the main drawbacks of the PGMs that govern EDAs is the lack of scalability and memory, since these models lose efficiency as the amount of solutions to learn from increases. Consequently, the common practice is to focus on the most promising solutions (in terms of fitness values) guiding the search towards that direction. DNNs can learn distributions from large datasets efficiently. However, using them for sampling solutions from these distributions to guide MEAs brings several particularities inherent in this kind of models to the EA. Studying these singularities is essential for reaching performance optimality and, in this case, we focus our efforts on DNN learning, and sampling using back-drive. Model learning involves (1) parameter initialization and (2) optimization. Performing back-drive also necessitates of (3) solution initialization. (4) The $y*$ required to the model in the back-drive procedure can also be configured in multiple manners. As the cost of performing inference with regression DNNs is considerably low, any possibility of performance improvement extracted from using the (5) DNN as a surrogate model is worth studying. For each of these five key features, two proposals are presented in the following sections. These pairs of proposals are later tested against each other in an extensive experimental section in an effort to determine the best conditions in which the back-drive can operate. The goal of this work is not giving exact guidelines on the setting of back-drive, but illustrating the directions which research should follow regarding this algorithm.

A. *Model initialization*

The first choice to be made is the initial value of the DNN parameters, θ. In the first generation, a straightforward procedure is to randomly initialize the parameters to be learned. However, for the subsequent iterations of the algorithm, θ can be reinitialized to random values again, or simply inherit the learned parameters from the previous generation.

Because the problem is constant during the whole evolution, it could be expected that some parameters in θ could be reutilized by the DNN across generations, especially those in the first layers. In the worst-case scenario, in which information cannot be transferred between generations, the weights optimized in the previous generation would not be worse than a new random initialization.

B. *Training the model*

As mentioned in Section II-B, the step previous to back-driving a network is training the network itself. In this work, the network is suited to approximate a continuous problem;

the mapping between the solutions and their corresponding fitness values. Two dynamics have been identified as plausible for performing this action.

The first one is based on the traditional manner in which MEAs operate; at each generation a fixed number of solutions are evaluated and used to train the model that will later be sampled. Commonly, MEAs maintain a fixed population size, replacing individuals as the evolution advances, in order to keep the learning task within reasonable computational cost.

The second variant is based on the fact that DNNs offer the best performance when large amounts of data are available. Therefore, solutions from past generations are retained, which results in more robust models after training.

C. Back-drive initialization

The individuals modified by the back-drive method need to be initialized too. Analogously to θ, in the first generation, random initialization is the straightforward choice. The following generations, though, place the question of whether the individuals should be reinitialized, or whether each model in each generation should take the individuals where the previous model left them off (an approach similar to [11]).

In this case, the algorithm faces an exploration vs exploitation trade-off scenario. Reinitializing the individuals in each generation would, intuitively, promote the creation of distributed populations. Using the individuals modified in the previous generation as a starting point before back-driving could favor focusing on certain areas of the search space.

D. Target value

Transforming every random input to an optimal solution could be an unrealistic objective for back-drive. Additionally, it could lead to some kind of homogeneity between solutions. These two scenarios could be avoided by introducing some level of noise in the target variable y^*, or directly by using a sub-optimal value. Moreover, the inclusion of other forms of the target variable, such as the logarithm, square, etc. could lead to a more informed back-drive procedure capable of generating solutions closer to the optimal fitness. These model-related questions do not add any computational cost to the algorithm, and could improve the results of the back-drive.

E. DNN as surrogate model

The fact that the back-drive is able to improve individuals does not necessarily mean that all of them reach the target value. Sometimes, back-drive does not work properly, depending, for example, on the individual initialization, or the way in which the DNN approximates the fitness function. Discarding individuals which are unlikely to benefit the evolutionary process before wasting evaluations is a key aspect of an efficient back-drive based EA. Reusing the same model employed for back-drive as a surrogate, it is possible to estimate the fitness value of all the back-driven solutions in order to keep the most promising ones. That is, generate a large number of back-driven individuals (which, with today's technology, does not result costly) and use the trained DNN to obtain an estimation of their fitness value. This way, the algorithm could avoid the evaluation of poor individuals, which would be especially beneficial when facing a real-world problem in which evaluations are highly costly. In EDAs, different models have been also used as surrogates to avoid the number of solutions evaluated [12], [30], [31].

F. Implementation

For these exploratory experiments, an MLP with two hidden layers is employed to learn the mapping between the solutions and their fitness value. The first layer is composed of as many neurons as three times the number of variables in a solution, whereas the second one contains twice the number of variables. The rest of the specification is as follows:

- ReLU activation function after the two hidden layers.
- Sigmoid activation function after the output layer (fitness values are scaled to [0-1] before learning).
- Weights (w_l) are initialized randomly, with Xavier initialization [32].
- Biases (b_l) are initialized to 0.

VI. Experiment setup

A. Problem benchmark

To determine what the optimal conditions for back-drive to operate in are, we have selected the widely known suite of CEC-2005 problems [33] as implemented in [34]. Despite this not being the most recent version of the problem suite, its extension is enough to test the different variants of the back-drive algorithm against each other. Two of the available functions (F7 and F25) have been discarded because the range of possible values of the variables is not fixed. Although the back-drive is able to operate in such conditions, we decided to discard the two functions for the sake of homogeneity. We adhere to the evaluation limit suggested in the benchmark for the number of variables $n = 10$: $100,000$ evaluations.

B. Model learning and back-driving

In order to learn the model, 25 epochs (divided in mini-batches of size 150) have been used in all cases. For optimization, the Adam [35] stochastic gradient descent technique was employed, with an initial learning rate of 0.001 in an offline manner (i.e., using the complete dataset instead of incrementally adding solutions as they are evaluated). So as to modify the individuals via back-drive, the same configuration of the gradient optimizer was used, only that, in this case, the learning was limited to just 500 epochs, which leads to very fast sampling.

C. Preliminary experimentation

In order to reduce the different variables in the evolutionary process and ease the analysis, we test the benefit of the model variants introduced in Section V-D in a vacuum, isolated from the evolutionary process, taking advantage of the reduced influence of the evolutionary component on this aspect. To that end, we learn a DNN with $10,000$ points, and back-drive it to obtain $1,000$ new points considering different experimental settings that allow the investigation of key questions about the behavior of the back-drive technique.

- Form of the fitness value (e.g., $log(f)$, f^2, etc.).
- Percentile of the fitness for back-driving. Requiring the back-drive to produce sub-optimal values could be beneficial for population quality or diversity.
- Noise: Whether the target value is the exact value for every sample, or has noise added to it. This could increase solution diversity.

TABLE II
MODEL LEARNING POSSIBILITIES TESTED (f = FITNESS VALUE).

f **transformations**	**Percentile**	**Noise**
$log(f)$	0 (best)	f
f	10	$abs(\mathcal{N}(f, 0.01))$
$\sqrt{f}$	20	
f^2	50	
$\sin(f)$		

The variants for the different aspects to be taken into account when learning the model, which, to the knowledge of the authors, have not been studied, are compiled in Table II. Note that the f transformations are accumulative, e.g., when requiring f^2, the three previous outputs are also included, mimicking a multiple regression scenario. Problem domains could be exploited with this technique, e.g., additively decomposable functions could have as many outputs as subfunctions.

By investigating these aspects in isolation, we can determine the best method for setting the target variable before the evolutionary process. For each combination, $1,000$ new random solutions were modified by back-drive and evaluated.

D. Analysis of the evolutionary process

Our next goal is to analyze the remaining components related to the use of back-drive, now within the EA framework. In the following, we enumerate these components and explain the two settings that have been investigated per aspect.

1) Model initialization: In order to show the benefits of the different approaches available for training the model to be used for back-drive, different tests are performed. With respect to θ initialization, two different evolutionary procedures have been carried off. In the first one, θ was randomly initialized in the first generation, and the rest of generations inherited it from its predecessor. In the second one, θ was randomly initialized in each generation.

2) Model training: With respect to the optimization of θ in each generation, the straightforward strategy of learning θ with the solutions in the last generation is used as a baseline. This approach is contrasted to a more sophisticated one in which information from the previous generations is retained:

- The first generation is retained in order to maintain diversity and provide perspective about bad solutions, i.e., give information to the model about undesired solutions.
- An elite population (of the same size as the rest of populations) with the best solutions found during the search is kept.
- At each generation, the dataset comprising the previous $k \leq 8$ populations is kept.

3) Back-drive initialization: Similarly to θ initialization, two EA configurations have been tested; reinitializing the individuals to be back-driven in each generation, or inheriting the individuals from previous generations.

4) DNN as surrogate model: We also test whether the prediction of the model for the modified individuals is a valuable source of information. Once the model has been trained, the cost of using back-driving on it is near constant with respect to the number of solutions modified. We propose a variant to the simple procedure of back-driving as many individuals as needed per generation: $20\times$ the number of individuals in a population are back-driven, and the best ones (as many as required for the population of the subsequent generation) are selected according to the fitness estimation given by the model.

VII. RESULTS

A. Preliminary experimentation

We investigate the three factors described in Section VI-C. An MLP is learned using a given dataset of solutions and the corresponding fitness values for a function F. Back-drive is used to sample new solutions from the model. Solutions are evaluated using the F function, and we use the fitness evaluation to assess the influence of each of the factors.

Fig. 1 summarizes the results of this experiment for the 24 functions in the benchmark. The left-hand side heatmap considers the three combinations which obtained the best results for each of the problems. The x axis shows the percentile value used as target for the back-drive algorithm. The y axis shows the different transformations of the target, and whether they had noise added. The number and color represent the number of times that combination was present in the top three performing combinations of the functions.

It can be clearly appreciated that requiring the model to produce the best solution (0% percentile) for each individual is the option that achieves the best results almost every time, whereas 10% hardly ever produces top-performing evolution, and 20% and 50% are not able to produce *good* results on any occasion. Regarding the addition of Gaussian noise to the target output, it is clear that requiring the exact fitness instead of perturbations of that value is also a better option. Finally, Fig. 1 shows that adding more outputs to the regression problem does not help the model to perform better back-drive, and, from three outputs on, the results are considerably worse.

The figure on the right-hand side of Fig. 1 shows, for each percentile (x axis) and function (y axis), the summation of the variances of the fitness values computed from the back-driven individuals. The summation is chosen in order to display more readable numbers. The goal of this figure is to determine whether requiring *worse* solutions to the back-drive results in larger diversity. It can be seen how, with few exceptions (F4, F6, F12, F13, F17, and F21), the variance is similar for all percentiles. Moreover, surprisingly, for other functions (F1, F5, F15, F16, F18, F19, F20 and F24) the variance is even greater

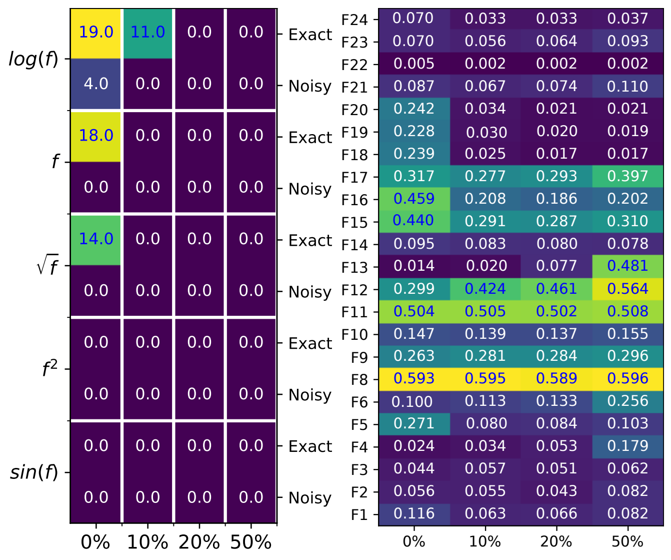

Fig. 1. Results of the preliminary experimentation. The figure on the left-hand side shows the frequency of appearance of the parameter combinations signaled by the x and double y axis, in the top-3 combinations for each problem. The other figure shows the variance found in the fitness value of the individuals generated via back-drive with different target percentiles, for each problem.

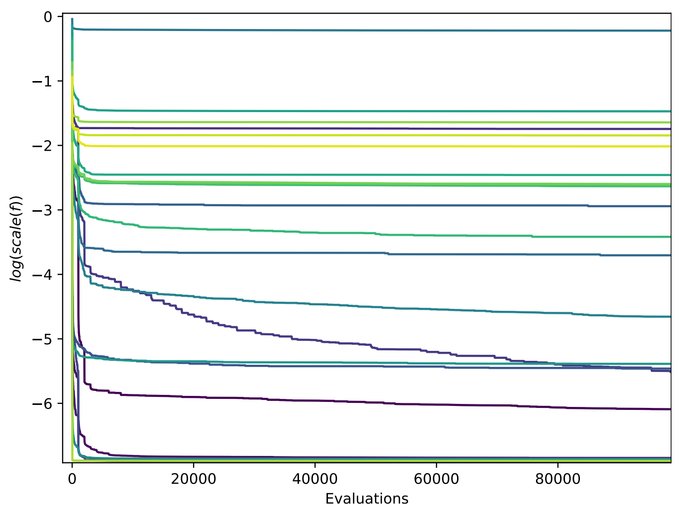

Fig. 2. Logarithm of the scaled fitness for each problem in the benchmark. The best fitness value found through the optimization process is shown.

when requiring *better* individuals. The rest of functions do not show such sizeable differences.

After studying these results, we can determine that simply using the exact logarithm of the fitness of the best solution provides better results in terms of quality and diversity.

B. Main experimentation

The characteristics of the model have been studied, and the most advantageous conditions for the back-drive to operate, discovered. It is now time to observe back-drive's behavior with different variations of evolutionary-bounded components.

With this goal in mind, an extensive set of experiments has been carried out. For each function in the benchmark all combinations of the back-drive described in Section VI-D have been performed. Each one of these processes was run 30 times. The EA consists of 100 generations of populations of size $1,000$.

Firstly, before going on to the analysis of the performance of the different formulations of the algorithm, we prove that the algorithm is indeed capable of improving individuals in terms of their fitness value. All fitness values generated by all 2 model training methods $\times 2$ DNN initialization methods $\times 2$ individual initialization methods $\times 2$ generation schemes $\times 30$ repetitions $= 4806$ were normalized to $(0, 1]$. After that, the mean of the best fitness value at each point during evolution of all the runs for each problem was computed. These averages are shown in Fig. 2. It can be seen how, as fitness evaluations are performed, the best found fitness value improves. There is great variety of convergence values for the different problems. The curves ending in higher (worse) values do so because few of the runs were able to reach much better fitness values than the rest. This way, when the normalization is performed, only these few values reach near-zero values, whereas the rest lie nearer to 1. Computing the mean of all runs derives in these higher fitness-valued lines.

Nevertheless, in most cases, evolution is satisfactory, as in general, the lines have a consistent decreasing slope.

We perform separate analysis of the different runs in order to devise the characteristics of the EA which makes the most of the back-drive algorithm.

As a first step, the fitness values at every point during the evolution of all 30 repetitions for each one of the 16 strategies are averaged. Next, for each evaluation in the averaged runs, the best registered fitness is recorded. Finally, the evolutionary component combinations present in the top-3 of the best fitness found after a given number of evaluations for each problem are awarded one point for that specific evaluating position. Therefore, at each position, $24 \times 3 = 72$ points are awarded. Fig. 3 shows this information, where the wider the stack, the more points a given component combination has obtained at different moments in the evolution.

In this figure, the common behavior of this kind of algorithm can be observed. In the first two generations the populations are still heavily affected by randomness and the variations of the back-drive have yet to impact the evolution. Once enough evaluations ($\sim 20,000$ or $\sim 20^{\text{th}}$ gen.) have been carried out, differences stabilize and it is easy to discern the component combinations which perform the best more frequently.

At a quick glance, one can observe that two colors in particular are found more frequently in the figure; green and yellow. These colors correspond with runs of the EA in which the DNN was trained with not only the information from the previous generation, but also incorporating knowledge from the initial random population and an elite population in which the best found individuals are stored, along with the individuals from the last eight generations. Due to the parity between the width of these two colors, it is possible to determine that modifying just as many individuals as the

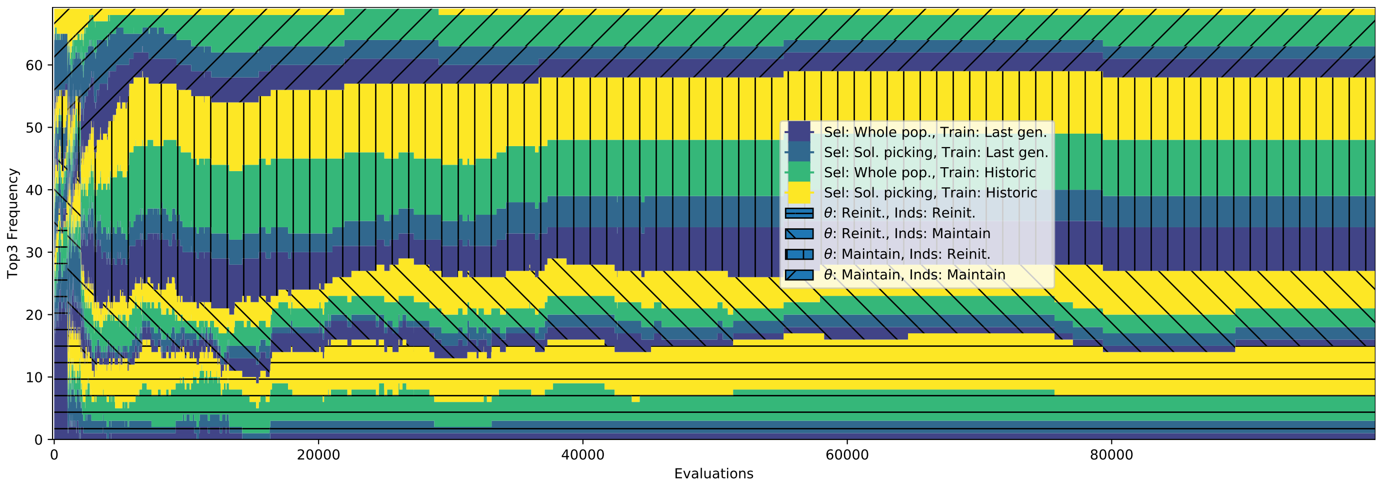

Fig. 3. Stackplot showing the frequency of appearance of each plot combination. The wider the line, the more present a parameter combination has been between the top three of each problem. Two individual selection methods are used, one in which the individual generation consists of as many individuals as required population, or creating $20\times$ the size of the population then using the DNN as a surrogate model to select the best ones and discard the rest. Two training methods are employed: One in which only the individuals from the last generations are used for learning θ, and the other in which solutions from past generations are kept for learning. θ can be reinitialized after each generation is completed, or it can be inherited between generations. Similarly, individuals changed by the back-drive can be inherited from the previous generation or randomly initialized each time.

population size or $20\times$ that number (and using the model as surrogate to select the most promising individuals) offer similar performances to the evolutionary process.

Additionally, regarding the hatches in the figure, the vertical lines are the most prominent ones. This indicates that using θ inheritance between generations is a very useful technique for the evolutionary procedure. Furthermore, it suggests that the individuals modified by the back-drive should not be initialized taking into account their values of the previous generation, but that they should be randomly initialized each time.

Other combinations, while also being present in varying degrees in the top-3 performing combinations, do not stand out as much as the combinations that have been highlighted.

VIII. Conclusions

In this paper, we have performed an in-depth analysis of the back-drive algorithm as the sole driving component of an evolutionary algorithm. Back-drive allows the usage of a DNN intended for supervised learning to learn the distribution of a dataset in order to sample more information from that distribution. The usage of DNNs in evolutionary algorithms could help avoid limitations traditionally present in model-based evolution. We have set the grounds of generally good practices to take into account when designing evolutionary algorithms based on back-drive.

Firstly, we have determined the structure of the network used to perform the back-drive, in terms of the output it produces. Because this characteristic is exclusive to the model, we have been able to perform this analysis isolated from the evolutionary process, which prevents the introduction of unnecessary noise. We have tested whether requiring the model to produce solutions with fitness values known to be suboptimal, requiring different transformations of these fitness values, or adding noise to them can have beneficial effects on the evolutionary algorithm. This analysis concluded that the trivial approach of model design, the one requiring a single form of the best fitness without noise, is the best option.

Once this fact has been confirmed, the different variations of the usage of the model in an evolutionary algorithm have been investigated. As the first step of this analysis, it has been proven that the back-drive technique is able to successfully drive an evolutionary algorithm in which individuals are improved regarding their fitness value.

Moreover, this second investigation concluded that the more information the model can get when being trained regarding previous generations, the better it will perform. Additionally, granting the model of the current generation access to the parameters learned by the model of the previous generation has proven to be another key for improved efficiency. Other variants which take advantage of the use of DNNs in evolutionary algorithms over more traditional approaches have not produced visible improvements.

A. Future work

To the knowledge of the authors, this is the first work in which the back-drive as a technique is used as the sole driver of an EA. The study performed in this paper is bounded to an initial investigation of the potential of this approach. In the following paragraphs, we consider future research lines.

In this work, the network layout has been fixed trough all the evolution and analysis. What is more, this structure was kept rather simple. Even though improving the structure does not imply performance improvement, intuition suggests it would. Moreover, enabling flexibility of the model which could adapt to the necessities of the algorithm at different phases of an evolutionary process (e.g., adding diversity vs. digging deeper into certain area of the search space) would most likely result in a major boost for the algorithm.

This work also uses a fixed population size. It is known that DNN models perform better with large amounts of data. More sophisticated management from the number of evaluations available could also be beneficial for the algorithm. For example, using a larger number of evaluations during the first stages of the evolution could help the algorithm to focus on a better area of the search space.

The way in which both the model learning and back-driving modifications are performed have also remained fixed during the whole of the evolutionary process. Parameter tweaking in this aspect would undoubtedly ensue improvements in the final outcome of the algorithm.

With respect to performance regarding time consumption, because the DNN training is performed based on mini-batches of solutions, a large part of the network training could be parallelized; as solutions are evaluated, they are fed to the learning algorithm while the rest of solutions are being evaluated. Model learning could be performed on-line, as opposed to the off-line method employed in this work.

Although the particular framework in which we have evaluated back-drive does not lead to a state-of-the-art algorithm, we reckon that back-drive could be an element to consider as part of these algorithms.

References

[1] D. Thierens, "The linkage tree genetic algorithm," *Parallel Problem Solving from Nature–PPSN XI*, pp. 264–273, 2011.

[2] R. Santana and A. Mendiburu, "Model-based template-recombination in Markov network estimation of distribution algorithms for problems with discrete representation," in *2013 Third World Congress on Information and Communication Technologies (WICT 2013)*. IEEE, 2013, pp. 170–175.

[3] S.-H. Hsu and T.-L. Yu, "Optimization by pairwise linkage detection, incremental linkage set, and restricted/back mixing: DSMGA-II," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, 2015, pp. 519–526.

[4] P. Larrañaga and J. A. Lozano, Eds., *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, 2002.

[5] J. A. Lozano, P. Larrañaga, I. Inza, and E. Bengoetxea, Eds., *Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms*. Springer, 2006.

[6] M. Pelikan, K. Sastry, and E. Cantú-Paz, Eds., *Scalable Optimization via Probabilistic Modeling: From Algorithms to Applications*, ser. Studies in Computational Intelligence. Springer, 2006.

[7] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, California: Morgan Kaufmann, 1988.

[8] D. Heckerman, D. Geiger, and D. M. Chickering, "Learning Bayesian networks: The combination of knowledge and statistical data," *Machine Learning*, vol. 20, pp. 197–243, 1995.

[9] M. Henrion, "Propagating uncertainty in Bayesian networks by probabilistic logic sampling," in *Proceedings of the Second Annual Conference on Uncertainty in Artificial Intelligence*, J. F. Lemmer and L. N. Kanal, Eds. Elsevier, 1988, pp. 149–164.

[10] M. Probst, "Generative adversarial networks in estimation of distribution algorithms for combinatorial optimization," *arXiv preprint arXiv:1509.09235*, 2015.

[11] S. Baluja, "Deep Learning for Explicitly Modeling Optimization Landscapes," *arXiv preprint arXiv:1703.07394*, 2017.

[12] U. Garciarena, R. Santana, and A. Mendiburu, "Expanding variational autoencoders for learning and exploiting latent representations in search distributions," in *Proceedings of the Genetic and Evolutionary Computation Conference*. Kyoto, Japan: ACM, 2018, pp. 849–856.

[13] A. Linden and J. Kindermann, "Inversion of multilayer nets," in *Proc. Int. Joint Conf. Neural Networks*, vol. 2, 1989, pp. 425–430.

[14] S. K. Pal and S. Mitra, "Multilayer perceptron, fuzzy sets, and classification," *IEEE Transactions on neural networks*, vol. 3, no. 5, pp. 683–697, 1992.

[15] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[16] D. E. Rumelhart, G. E. Hinton, R. J. Williams, and others, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[17] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[18] P. Larrañaga and J. A. Lozano, *Estimation of distribution algorithms: A new tool for evolutionary computation*. Springer Science & Business Media, 2001, vol. 2.

[19] A. W. Churchill, S. Sigtia, and C. Fernando, "Learning to generate genotypes with neural networks," *CoRR*, vol. abs/1604.04153, 2016. [Online]. Available: http://arxiv.org/abs/1604.04153

[20] M. Probst, F. Rothlauf, and J. Grahl, "Scalability of using restricted Boltzmann machines for combinatorial optimization," *European Journal of Operational Research*, vol. 256, no. 2, pp. 368–383, 2017.

[21] M. Probst, "Denoising autoencoders for fast combinatorial black box optimization," in *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, 2015, pp. 1459–1460.

[22] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, "BOA: The Bayesian optimization algorithm," in *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-1999*, vol. I. Orlando, FL: Morgan Kaufmann Publishers, San Francisco, CA, 1999, pp. 525–532.

[23] M. Probst and F. Rothlauf, "Deep Boltzmann machines in estimation of distribution algorithms for combinatorial optimization," *CoRR*, vol. abs/1509.06535, 2015. [Online]. Available: http://arxiv.org/abs/1509.06535

[24] B. T. Zhang, , and S. Y. Shin, "Bayesian evolutionary optimization using Helmholz machines," in *Parallel Problem Solving from Nature, Lecture Notes in Computer Science*, vol. 1917. Springer, 2000, pp. 827–836.

[25] L. Marti, J. García, A. Berlanga, and J. M. Molina, "Introducing MONEDA: scalable multiobjective optimization with a neural estimation of distribution algorithm," in *Proceedings of the 10th annual conference on Genetic and evolutionary computation GECCO-2008*. New York, NY, USA: ACM, 2008, pp. 689–696.

[26] H. Tang, V. Shim, K. Tan, and J. Chia, "Restricted Boltzmann machine based algorithm for multi-objective optimization," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE, 2010, pp. 1–8.

[27] V. A. Shim, K. C. Tan, C. Y. Cheong, and J. Y. Chia, "Enhancing the scalability of multi-objective optimization via restricted Boltzmann machine-based estimation of distribution algorithm," *Information Sciences*, vol. 248, pp. 191–213, 2013.

[28] A. W. Churchill, S. Sigtia, and C. Fernando, "A denoising autoencoder that guides stochastic search," *CoRR*, vol. abs/1404.1614, 2014. [Online]. Available: http://arxiv.org/abs/1404.1614

[29] S. Baluja, "Deep learning for explicitly modeling optimization landscapes," *CoRR*, vol. abs/1703.07394, 2017. [Online]. Available: http://arxiv.org/abs/1703.07394

[30] A. E. Brownlee, O. Regnier-Coudert, J. A. McCall, S. Massie, and S. Stulajter, "An application of a GA with Markov network surrogate to feature selection," *International Journal of Systems Science*, vol. 44, no. 11, pp. 2039–2056, 2013.

[31] R. Santana, A. Mendiburu, and J. A. Lozano, "Critical issues in model-based surrogate functions in estimation of distribution algorithms," in *Proceedings of the 4th Conference on Swarm, Evolutionary, and Memetic Computing (SEMCCO-2013)*, ser. Lectures Notes in Computer Science. Chennai, India: Springer, 2013, pp. 1–13.

[32] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 249–256.

[33] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y. P. Chen, A. Auger, and S. Tiwari, "Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization," Nanyang Technological University, Singapore, Tech. Rep., 2005.

[34] S. Wessing, "Optproblems: Infrastructure to define optimization problems and some test problems for black-box optimization," *Python package version 0.9*, 2016.

[35] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.