Denoising Autoencoders for Fast Combinatorial Black Box Optimization

Malte Probst
University of Mainz
Dept. of Information Systems and Business Administration
Mainz, Germany
probst@uni-mainz.de

ABSTRACT

We integrate a Denoising Autoencoder (DAE) into an Estimation of Distribution Algorithm (EDA) and evaluate the performance of DAE-EDA on several combinatorial optimization problems. We asses the number of fitness evaluations and the required CPU times. Compared to the state-of-the-art Bayesian Optimization Algorithm (BOA), DAE-EDA needs more fitness evaluations, but is considerably faster, sometimes by orders of magnitude. These results show that DAEs can be useful tools for problems with low but non-negligible fitness evaluation costs.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Neural Networks*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

Keywords

Autoencoder; Estimation of Distribution Algorithms; Combinatorial Optimization Problems; Neural Networks

1. INTRODUCTION

EDAs are metaheuristics for combinatorial and continuous non-linear optimization. They improve a population of solutions over consecutive generations [5]. In each generation, they approximate the dependency structure between the decision variables using a probabilistic model and use it to sample new candidate solutions. By repeated model estimation, sampling, and selection, EDAs can solve difficult optimization problems.

We integrate a DAE [12], a special type of neural network, as EDA model. We assess its performance on multiple standard benchmark problems from combinatorial optimization and include results for BOA [8] for comparison.

2. AUTOENCODERS

An Autoencoder (AE) AE has a visible layer $x \in [0,1]^n$, a hidden layer $h \in [0,1]^m$, and an output layer $z \in [0,1]^n$, which are connected by two deterministic functions: the encoding function $h = c(x;\theta)$ and the decoding function $z = f(h;\theta')$, with parameters θ, θ'. The training objective of the AE is to find parameters θ, θ' which minimize the *reconstruction error* $Err(x,z)$, i.e., the difference between x and z for all examples $x^i, i \in (1,\dots,\tau)$ in the training set:

$$\theta, \theta' := \operatorname*{argmin}_{\theta,\theta'} \frac{1}{\tau} \sum_{i=1}^{\tau} \mathrm{Err}(x^i, z^i). \tag{1}$$

A common choice for $\mathrm{Err}(x,z)$ is the cross entropy function $\mathrm{Err}(x,z) = -\sum_{k=1}^{n}[x_k * \log(z_k) + (1-x_k) * \log(1-z_k)]$, encoding and decoding functions are usually chosen as $c(x) = \mathrm{sigm}(x * W + b_h)$ and $f(h) = \mathrm{sigm}(h * W' + b_z)$, where $\mathrm{sigm}(x) = \frac{1}{1+e^{-x}}$ is the logistic function, W and W' are weight matrices of size $(n \times m)$ and $(m \times n)$, respectively, and $b_h \in \mathbb{R}^m$, $b_z \in \mathbb{R}^n$ are biases which work as offsets. Often, W and W' are *tied*, i.e., $W' = W^\top$. Then, the AEs configurable parameters are $\theta = \{W, b_h, b_z\}$.

Minimizing (1) is performed by using stochastic gradient descent (SGD) algorithm (see Algorithm 1).

Algorithm 1 Pseudo code for training an Autoencoder

1: **Initialize** $\theta = \{W, b_h, b_z\}$ randomly, set $0 < \alpha < 1$
2: **while** not converged **do**
3:    **for each** example i in the training set **do**
4:       $\theta := \theta - \alpha * \frac{\delta \mathrm{Err}(x^i,z)}{\delta\theta}$, with $z = f(c(x^i;\theta);\theta)$
5:    **end for**
6: **end while**
7: (for training a DAE, replace x^i with $q(\hat{x}^i|x^i)$ in line 4)

Algorithm 2 Pseudo code for sampling a DAE

1: **Given** $\theta = \{W, b_c, b_f\}$, $c(\cdot)$, $f(\cdot)$, $q(\cdot)$
2: **Initialize** $x \in [0,1]^n$ randomly
3: **for a fixed number** s **of sampling steps do**
4:    $x := z = f(c(q(\hat{x}|x);\theta);\theta)$
5: **end for**
6: Use x as a sample from the DAE

If m is large enough, a trivial way to solve (1) is to learn the identity function where each x_i is directly mapped to the corresponding z_i. A *Denoising* AE forces the model to learn a more useful representation, using regularization [12]. Each training example x is corrupted by a stochastic mapping $\hat{x} = q(\hat{x}|x)$, i.e., we add random noise. The DAE then calculates the reconstruction of the corrupted input as $z = f(c(\hat{x};\theta);\theta)$. The parameters are updated in the direction of $\frac{\delta \mathrm{Err}(x,z)}{\delta\theta}$. Hence, the DAE tries to reconstruct x rather than $\hat{x}$. Samples can be generated from the DAE using the process proposed in [1] (see Algorithm 2). Note

GECCO '15 July 11-15, 2015, Madrid, Spain

ACM ISBN 978-1-4503-3488-4/15/07.
DOI: http://dx.doi.org/10.1145/2739482.2764691

Problem	Algorithm	Average results Population size such that optimum is found in ≥90% of runs Evaluations	 Time (sec)
4-Traps 20 bit	BOA	3,150±391	189±31
	DAE-EDA	3,075±1,798	**52***±17
4-Traps 40 bit	BOA	**11,350***±1,195	2,833±466
	DAE-EDA	23,100±6,292	**185***±33
4-Traps 60 bit	BOA	**18,250***±1,445	10,811±1,165
	DAE-EDA	41,050±2,783	**449***±37
5-Traps 25 bit	BOA	**9,550***±921	951±125
	DAE-EDA	14,325±4,611	**87***±17
5-Traps 50 bit	BOA	44,333±2,357	19,866±1,610
	DAE-EDA	37,500±11,033	**332***±74
5-Traps 75 bit	BOA	108,000±5,692	114,337±7,005
	DAE-EDA	**57,300***±4,529	**871***±76
NK $n=30$, $k=4$, $i=1$	BOA	25,200±3,929	3,913±875
	DAE-EDA	26,175±6,495	**181***±47
NK $n=30$, $k=4$, $i=2$	BOA	**66,800***±14,593	12,726±3,500
	DAE-EDA	260,400±70,494	**1,089***±308
NK $n=34$, $k=4$, $i=1$	BOA	**20,700***±3,378	3,461±640
	DAE-EDA	50,000±13,431	**313***±88
NK $n=34$, $k=4$, $i=2$	BOA	58,950±10,230	11,974±2,151
	DAE-EDA	**30,650***±12,285	**298***±85
NK $n=30$, $k=5$, $i=1$	BOA	**12,450***±2,274	1,582±341
	DAE-EDA	100,200±23,110	**499***±166
NK $n=30$, $k=5$, $i=2$	BOA	**54,450***±6,168	8,762±1,503
	DAE-EDA	75,000±10,941	**348***±61
NK $n=34$, $k=5$, $i=1$	BOA	242,400±35,517	64,622±11,539
	DAE-EDA	**73,950***±17,932	**380***±114
NK $n=34$, $k=5$, $i=2$	BOA	271,200±57,349	74,570±20,228
	DAE-EDA	**179,100***±63,591	**749***±307
HIFF64	BOA	**11,825***±1,477	7,025±1,029
	DAE-EDA	19,450±2,247	**324***±31
HIFF128	BOA	**39,350***±3,410	93,144±10,542
	DAE-EDA	56,750±5,421	**2,624***±151

Table 1: This table shows average values for fitness evaluations and CPU time for DAE-EDA, and BOA for the test problems. For each instance and algorithm, we selected the minimal population size which lead to the optimum in ≥ 90% of the runs. Results are averaged over 20 runs. Results marked with (*) are significantly smaller, according to pairwise Wilcoxon signed-rank tests ($p < 0.01$, data is not normally distributed)

that each sample is a vector $x \in [0,1]^n$. To turn this vector of real-valued elements into a candidate solution for the EDA, i.e., a binary string, we sample each variable x_i from a Bernoulli distribution with $p = x_i$.

3. EXPERIMENTAL SETUP

We use several instances from the standard benchmark problems concatenated deceptive traps [2], NK landscapes [3] and the HIFF function [13]. All three problems are composed of subproblems, which are either deceptive (traps), overlapping (NK landscapes), or hierarchical (HIFF), and therefore multimodal. For each instance and algorithm, we test 20 runs of popsize $\in \{50; 100; \ldots; 16,000\}$. In each run, the EDA is allowed to run for 100 generations and terminates, if there is no improvement of the best solution for more than 20 generations. We use tournament selection without replacement of size two [6]. For the DAE, we choose $m = n$, $s = 10$, and $\alpha = 0.2$. The corruption process $q(\hat{x}|x)$ randomly corrupts 10% of the inputs by setting them to 0 or 1. The batch size for SGD is $b = 100$. We apply the simple parameter control scheme from [11] to determine when to stop DAE training.

All algorithms are implemented in Matlab/Octave and executed using Octave V3.2.4 on a on a single core of an AMD Opteron 6272 processor with 2,100 MHz.

4. RESULTS AND CONCLUSION

For each problem instance and algorithm, we select the minimal population size which leads to the optimum in ≥ 90% of the runs. We report the average number of fitness evaluations and CPU time of those runs (see table 3).[1] As expected, BOA has the better overall performance in terms of fitness evaluations. However, most of the time the number of fitness evaluations required by DAE-EDA is in the same order of magnitude. The results suggest that DAE-EDA is able to decompose the test problems properly, and solve the parts independently. For all but one instance, DAE-EDA is significantly faster than BOA, sometimes by multiple orders of magnitude. This is due to the much quicker model building and sampling of the DAE. Note that the direct comparison of CPU times is not entirely fair for BOA, due to the script-based programming language. However, most recent implementations of neural networks are parallelized on graphics processing units (GPU), yielding high speedups (see e.g. [4]). Accordingly, in the optimization context, parallelizing a neural EDA model can yield very high speedups, compared to other parallelizations [10, 7].

In sum, DAE-EDA can be a useful tool for solving complex combinatorial optimization problems, where fitness evaluation costs are low, but non-negligible.

5. REFERENCES

[1] Y. Bengio, L. Yao, G. Alain, and P. Vincent. Generalized Denoising Auto-Encoders as Generative Models. In *Advances in Neural Information Processing Systems 26 (NIPS'13)*. NIPS Foundation (http://books.nips.cc), 2013.

[2] K. Deb and D. E. Goldberg. *Analyzing Deception in Trap Functions*. University of Illinois, Department of General Engineering, 1991.

[3] S. A. Kauffman and E. D. Weinberger. The NK Model of Rugged Fitness Landscapes and its Application to Maturation of the Immune Response. *Journal of theoretical biology*, 141(2):211–245, 1989.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet Classification with Deep Convolutional Neural Networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[5] P. Larrañaga and J. A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation.* Genetic Algorithms and Evolutionary Computation, 2. Kluwer Academic Pub, 2002.

[6] B. L. Miller and D. E. Goldberg. Genetic Algorithms, Tournament Selection, and the Effects of Noise. *Complex Systems*, 9:193–212, 1995.

[7] J. Očenášek and J. Schwarz. The Parallel Bayesian Optimization Algorithm. In *The State of the Art in Computational Intelligence*, pages 61–67. Springer, 2000.

[8] M. Pelikan. Bayesian Optimization Algorithm. In *Hierarchical Bayesian Optimization Algorithm*, volume 170 of *Studies in Fuzziness and Soft Computing*, pages 31–48. Springer Berlin / Heidelberg, 2005.

[9] M. Probst. Denoising Autoencoders for Fast Combinatorial Black Box Optimization. *preprint on arXiv*, arXiv:1503.01954, 2015.

[10] M. Probst, F. Rothlauf, and J. Grahl. An Implicitly Parallel EDA Based on Restricted Boltzmann Machines. In *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, GECCO '14, pages 1055–1062, New York, NY, USA, 2014. ACM.

[11] M. Probst, F. Rothlauf, and J. Grahl. Scalability of Using Restricted Boltzmann Machines for Combinatorial Optimization. *preprint on arXiv*, abs/1411.7542, 2014.

[12] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and Composing Robust Features with Denoising Autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[13] R. A. Watson, G. S. Hornby, and J. B. Pollack. Modeling Building-Block Interdependency. In *Parallel Problem Solving from Nature - PPSN V*, pages 97–106. Springer, 1998.

[1] For the more results including a univariate EDA, a neural network based EDA and a DAE-based local search see [9]