A Neural Network that Guides Stochastic Search

Alexander W. Churchill, Siddharth Sigtia, and Chrisantha Fernando

School of Electronic Engineering and Computer Science
Queen Mary, University of London
{a.churchill,s.s.sigtia,c.t.fernando}@qmul.ac.uk

Abstract. Typically in a genetic algorithm, the variability operators, i.e. the way that a parent generates offspring, is not modified during the evolutionary run. Here we adaptively learn an exploration distribution by training a denoising autoencoder online at each generation to reconstruct a slowly decaying memory of the best genotypes so far encountered. By using a compressed hidden layer, the autoencoder is forced to learn hidden features in the training set that may generalise to novel problems. It's output neurons define a probability distribution that we sample from to produce offspring solutions. The algorithm outperforms a canonical genetic algorithm on several combinatorial problems, including the multidimensional 0/1 knapsack problem, MAXSAT and the Hierarchical If and Only If, and on some parameter optimisation algorithms such as the Rastrigin and Rosenbrock functions. The neural network is able to learn hidden structure in the fittest genotypes, whilst sampling from the network using a Bernoulli distribution or with Gaussian noise maintains diversity, enabling optimal solutions to be found on NP-hard problems.

1 Introduction

Evolutionary Algorithms typically employ static exploration methods, such as recombination and mutation operators, in order to traverse a search space. A problem with this approach is that 'building blocks', structural features of the search space, can be easily broken, discarding important information acquired during the search process. Addressing this problem are Estimation of distribution algorithms (EDAs), which attempt to statistically model sections of a search space in order to uncover underlying structure and guide search towards optimal solutions in an efficient manner [1]. In addition to accumulating structural information about a problem instance during task, a natural extension of model-building algorithms would be to accumulate and exploit knowledge between tasks. This opportunity appears to have been ignored by the EDA community.

At the heart of any EDA lies a model-building process. Examples include Bayesian Networks, Markov Networks and K-Means clustering [2]. In this paper we introduce a novel neural-based method for modelling: a denoising autoencoder. An autoencoder is a feed forward neural network, consisting of at least one hidden layer, which is trained to reproduce its inputs from its outputs. Over

the course of training, the hidden layer forms a compressed representation of the inputs, which can be used for various machine learning tasks [3]. We claim that a denoising autoencoder can be used to learn complex probability distributions that define the best genotypes in a generation, and guide search towards promising regions. Results show that an autoencoder based optimisation algorithm is able to outperform a canonical genetic algorithm (GA) across a range of combinatorial optimisation and parameter optimisation problems. An advantage of the autoencoder is that it is trained online, allowing it to be presented with examples from different problem instances. In a simple example we show that this method is able to learn structure between two instances of a problem class to speed up convergence on the optimal solution of a third instance, highlighting the potential of the autoencoder to accumulate knowledge.

2 Background

EDAs (also known as Probabilistic-Model-Building Genetic Algorithms) are population based optimisers, that typically replace genetic operators with a statistical model. The rationale behind the model building and related linkage learning approach is that dependencies between variables can be captured and preserved. Early work on EDAs concentrated on methods that explicitly modelled the probabilities of features occurring independently in a population of genotypes. These include the compact Genetic Algorithm [6], PBIL [4] and Univariate Marginal Probability methods [2]. Improved success was found by modelling multivariate dependencies using clustering algorithms (ECGA) [5], Bayesian Networks, Markov Networks and tree structures [2], among others.

Our use of the autoencoder model is motivated by its potential to learn high-dimensional non-linear dependencies in data, while maintaining a low computational cost in terms of training time. Recently there has been interest in neural-based methods in an EDA context for multi-objective optimisation. A Growing Neural Gas (GNG) was used as a model in [7], employing a competitive Hebbian learning rule to cluster data without having to pre specify the number of groups. A shallow Restricted Boltzmann Machine (RBM) was used to model high dimensional data in [8], beating the state-of-the-art on several multi-objective continuous benchmark problems. An autoencoder is another neural-based method for the unsupervised learning of features and has hitherto not been applied to combinatorial or continuous optimisation.

A second motivation for this approach is to investigate methods in which evolutionary algorithms can be implemented using neural structures. The *Neural Replicator Hypothesis* [9] proposes that evolutionary dynamics could operate in the brain at ontogenetic timescales. In [9], a (1+1)-ES is implemented in a network of spiking neurons. Adding Hebbian learning enabled linkages to be found between features, and the 128-bit HIFF problem to be solved significantly faster than by a genetic algorithm.

3 Methods

3.1 The Autoencoder

In this paper, we present a system that uses a denoising autoencoder (dA) to model the structure of the best solutions at each generation. New solutions at each generation are produced by sampling from the dA. Therefore, the two primary uses of the dA are for model building and sampling new solutions.

The population of good solutions at each generation can be assumed to be a sample from a data generating distribution. Data generating distributions for high-dimensional data often have complex structure that cannot be modelled accurately by simple probabilistic models. Models such as RBMs, Bayesian nets and Helmholtz machines have been used in the past to this end [10, 8, 11].

A standard autoencoder consists of an encoder and a decoder. The encoder performs an affine transformation followed by an element-wise non-linear operation. The mapping performed by the encoder is deterministic and can be described as:

$$h_\theta(\mathbf{x}) = f(\mathbf{W}\mathbf{x} + \mathbf{b})$$

The decoder is also a deterministic network and operates on the encoded input $h_\theta(x)$ to produce the reconstructed input:

$$r_{\theta'}(\mathbf{h}) = g(\mathbf{W}'\mathbf{h} + \mathbf{b}')$$

The autoencoder learns the distribution of the training data by means of the layer of hidden variables. The encoder network is a non-linear generalisation of PCA [3]. The autoencoder learns interactions between the input attributes and maps it to the hidden layer via a non-linear transformation, making it a powerful model for learning and exploiting structure present in the best individuals.

We interpret the outputs of the decoder network as parameters for a conditional distribution $p(X|Z = \mathbf{z})$ over the outputs of the network given an input $\mathbf{z}$. For binary optimization problems, the outputs $\mathbf{z}$ are considered to be parameters for the distribution $X|\mathbf{z} \sim \mathcal{B}(\mathbf{z})$ where $\mathcal{B}$ is the bernouilli distribution. For continuous parameter-optimisation problems, the outputs $\mathbf{z}$ parameterise a multi-variate normal distribution $X \sim \mathcal{N}(\mathbf{z}, \sigma^2)$, where the covariance matrix is assumed to be diagonal and the standard deviation along the diagonal is a tuneable parameter.

In our algorithm we use a denoising autoencoder (dA) which is a variant of the standard autoencoder [12]. The dA tries to recreate the input $\mathbf{x}$ from 'corrupted' or 'noisy' versions of the input, which are generated by a stochastic corruption process $\tilde{\mathbf{x}} = \mathbf{q}(\tilde{\mathbf{x}}|\mathbf{x})$. Autoencoders are trained along with strong regularisation in order to impose an information bottleneck on the model. If this is not done carefully, it would be very easy for the autoencoder to learn the identity transformation which would not be very useful. Adding the denoising criterion to the model forces the autoencoder to learn more robust representations which are invariant to perturbations of the input. Although denoising was introduced in order to encourage the learning of more robust representations, in

our system we employ the denoising criterion to widen the basins of attraction around the best individuals in every generation. During training, we train the autoencoder on the most promising solutions, and by increasing corruption we encourage individuals that are far away from the training set to move towards the nearest high quality solution. By making use of the corruption noise as a tuneable parameter, the extent of the basins of attraction can be controlled, with high corruption giving rise to large basins of attraction.

We present a toy example in order to validate our hypothesis. We train an autoencoder on a 6-bit problem where the target vectors are all ones, all zeros, and the 3 least significant bits set to 1. Fig 2 shows matrix of transition probabilities for all possible pairs of genomes. There are 64 possible genomes for the 6-bit problem. The indices of the X and Y axes are labelled as the sequence of binary numbers: the 0-index corresponds to the string of all 0s and the 63rd index corresponds to the string of all ones. Each point (a,b) in the figure denotes the probability of transition from genone a to b. From the figure we observe that there are bands of high probabilities along $Y = 0, 7, 63$. This implies that for most input genomes, the autoencoder has a high probability of producing a genome that is close to one of the target solutions. In figure 3, we plot the marginalised probabilities of each of the target genomes. We observe peaks at the location of the three target vectors. We also notice that there is a band of high probability between the vectors 000000 and 000111. This demonstrates that the autoencoder learns the structure of the best solutions and yields outputs that are closer to the best solutions. The fact that the new population has probability peaks around all three target solutions demonstrates that the autoencoder learns a complex non-linear mapping from input to output and is capable of retaining information about several classes of solutions, thus enabling more effective search.

We perform another experiment to test the effect of corruption noise on the outputs of the dA. We argue that increasing corruption noise widens the basin of attraction around the training examples. In this experiment, we train the GA on a 20-bit MaxOnes problem. The choice of the problem was motivated by the fact that MaxOnes has a unique optimum solution and the hamming distances from the optimal solution are easy to interpret. We allow the GA to run for 10 iterations and train the autoencoder on the best solutions at every iteration. The inputs to the dA are corrupted by a binary corruption process that stochastically flips a fixed proportion of bits for each example. The autoencoder is then trained to reconstruct the true targets from the corrupted inputs. After 10 iterations we present a set of randomly generated input vectors and sample a new population from the dA. We repeat this experiment with different corruption rates and the same set of input vectors and plot the hamming distances of the new population from the optimum solution. From figure 3 we observe that increasing the corruption rate leads to a decrease in the hamming distances. This shows that as corruption noise is increased, the outputs of the dA are more likely to be closer to an example in the training set. ?

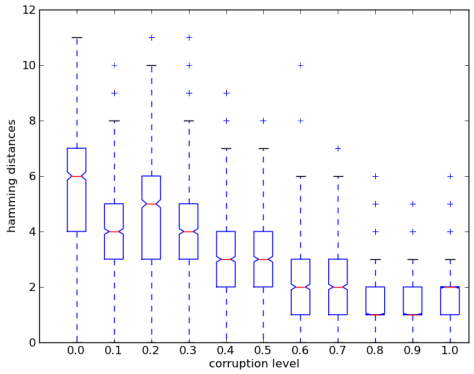


Fig. 1: Hamming distances to optimal solution on MaxOnes with different corruption levels.

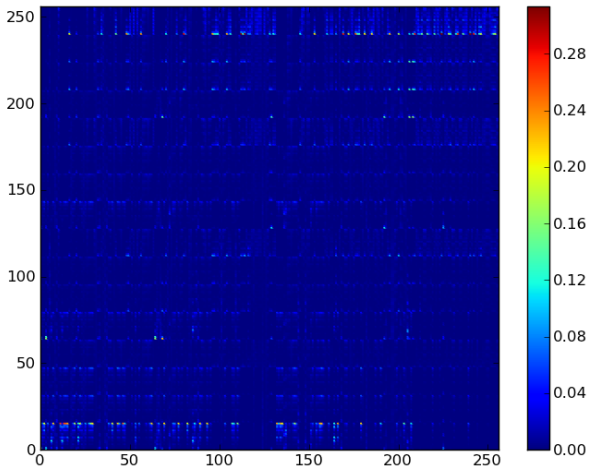


Fig. 2: 8-bit HIFF transition matrix.

3.2 Optimisation Algorithm

The pipeline of the *Denoising Autoencoder Genetic Algorithm* (DAGA) is similar to other EDAs and is inspired by methods in HBOA [13]. A population of solutions is maintained, P_t, and updated at each iteration, t. An initial population of solutions, P_0, is drawn from a uniform distribution. These solutions are evaluated and the fittest unique x% are selected (i.e. truncation selection) to be in the training set, $\hat{P}_t$. The Denoising Autoencoder, D_t, is then trained with $\hat{P}_t$

for e epochs. Following training, a new set of solutions, S_t, is selected from P using tournament selection (with replacement). Each member of S_t is inputted to D_t, and the output vector, y, is sampled from using a binomial distribution, to produce a new solution. This solution is included in P_{t+1} if it is better than its closest neighbour according to Restricted Tournament Selection (RTR) [13]. Pseudocode for DAGA is presented in Algorithm 1.

4 Experiments

DAGA is tested on three sets of experiments: (1) discrete, (2) continuous and (3) accumulation of adaptation problems.

4.1 Discrete

We perform experiments on two different instances of the **Multi-dimensional Knacksack problem**. The first is the Weing8 instance [14], which has 105 items and two constraints (optimal solution is 602,319), and the second is a randomly generated instance with 500 items and one constraint (optimal solution is 10,104). Both instances are available in the supplementary material.

The proposed system is then applied to the 128-bit and the 256-bit **Hierarchical If and Only If problem (HIFF)** [15]. In order to remove any possible bias towards an all 1s solution, solutions from DAGA have a random bit mask (fixed before each trial) applied to them before evaluating fitness.

We also test the system on a **128-bit Royal Road problem** with 8-bit partitions [16]. As with the HIFF, a random mask is applied to the output of DAGA before fitness evaluation.

Finally, the system is tested on a **3-CNF, 100-bit** MAXSAT problem obtained from SATLIB [17]. The problem consists of 430 clauses and belongs to the phase transition region, which is the point at which the problem transitions from generally solvable to generally unsolvable.

4.2 Continuous

We evaluate our system's performance on continuous parameter optimisation problems by applying it to a 50-d Sphere, a 10-d Rosenbrock and a 10-d Rastrigin function. The Sphere function is given by $f(p) = \sum_{i=1}^{n} p_i^2, -5.12 < p_i < 5.12$. The Rosenbrock function is $f(p) = \sum_{i=1}^{n} 100(p_{i+1} - p_i^2)^2 + (1 + p_i)^2, -2.048 \leq p_i \leq 2.048$. And finally the Rastrigin function is defined as $f(p) = 10.n + \sum_{i=1}^{n} \left[p_i^2 - 10.\cos(2\pi p_i)\right], -5.12 \leq p_i \leq 5.12$.

4.3 Accumulation of Adaptation

We investigate whether dAGA can be used to achieve faster convergence on new tasks, that are similar to tasks that it has already learnt to solve. In order to test the validity of this claim, a new task was devised that was influenced by Watson

et al [18]. dAGA is applied to solve three problem instances in sequence, without reinitialising the weights of the dA between instances. The problems consist of minimising the hamming distance to a target string, which represents an 81-pixel image. The algorithm is evaluated on the speed up created by solving the third solutions, where speed up is defined as the difference in the number of iterations needed to solve the third problem after solving the first two, compared to solving the third problem on its own. The first and second solutions are a Cross and Box pattern. The third solution is a pattern made by recombining quarters from the original patterns. As each new solution can inherit a quarter from either the Cross or the Box pattern, there are 2^4 unique solutions. dAGA is tested on training on the Cross and the Box (XB) and then on one of the 16, as well as the Box and the Cross (BX) and then on one of the other combined patterns, leading to 32 experiments.

5 Results

On the discrete and continuous problems, dAGA is compared to a canonical generational Genetic Algorithm (GA). On the continuous problems it is additionally compared to a (1+1)-ES with the 1/5 rule [**?**]. The GA employs tournament selection, two point crossover (with probability p_c) and on the discrete problems there is a probability, p_m, of a bit flip mutation at each allele, while on the continuous problems there is probability, p_m, of adding Gaussian noise to each component of the vector from $N(0, \delta)$. Programming code is available in the online supplementary material. For each experiment a large parameter sweep was performed on both dAGA and the GA and the best found configurations were chosen for comparison. Due to the stochastic nature of the algorithms each experiment is repeated 10 times.

5.1 Discrete

Table **??** presents details of the best solution returned at the end of a run and the number of evaluations required for dAGA and the GA on the 6 problems described above in 4.1. The table shows that dAGA finds either significantly better solutions, or optimal solutions significantly faster than the GA, on all of the problems apart from the Royal Road where they are not significantly different. On four of the six experiments (both knapsacks, MAXSAT and 256-bit HIFF), dAGA finds significantly better solutions than the GA, within the given evaluation limits. On the Weing8 knapsack instance dAGA reaches the optimum solution 8 out of 10 attempts, and on the 500-item instance 7 out of 10 attempts. On the MAXSAT instance, dAGA also reaches the optimum 70% of the time. On all three of these problems the GA is unable to locate the optimal solution a single time. On the 128-bit HIFF, dAGA locates the optimal solution every time, while the GA locates it on only half of the trials and on the 256-bit HIFF dAGA again has a 100% success rate while the GA achieves only 30%. On the Royal Road, dAGA locates the global optima on every trial, while the GA finds it on 9

Experiment	Algorithm	Min	Max	Mean	Mean Evals	Success %
MAXSAT	GA	424.0	429.0	426.5 ±1.4	500000.0 ±0.0	0%
	AE	429.0	430.0	429.8 ±0.4*	291944.4 ±134968.0*	70%
128 HIFF	GA	832.0	1024.0	940.8 ±86.1	416000.0 ±87430.0	50%
	AE	1024.0	1024.0	1024.0 ±0.0	231000.0 ±23537.2*	100%
256 HIFF	GA	1664.0	2304.0	1984.0 ±225.4	1590500.0 ±626719.4	30%
	AE	2304.0	2304.0	2304.0 ±0.0*	1355500.0 ±190793.7	100%
Knapsack Weing8	GA	619568.0	621086.0	620099.2 ±529.8	100000.0 ±0.0	0%
	AE	621086.0	624319.0	623124.3 ±1416.6*	96600.0 ±34414.2	80%
Knapsack 500	GA	10047.0	10093.0	10074.2 ±14.0	200000.0 ±0.0	0%
	AE	10096.0	10104.0	10102.7 ±2.5*	121620.0 ±52114.9*	70%
Royal Road	GA	120.0	128.0	127.2 ±2.4	42600.0 ±27034.9	90%
	AE	128.0	128.0	128.0 ±0.0	26500.0 ±3721.6	100%

Table 1: Results for dAGA and the GA on the 5 problems. Showing the value of the minimum and maximum solution returned at the end of search, and the mean best solution in the population after 25%, 50%, 75% and 100% of evaluations had been completed, across 10 trials. Stars indicate that the mean result for dAGA is significantly different from the GA according to a Wilcoxan Rank Sum test ($p < 0.05$).

out of 10. These results show that on a wide range of discrete problems dAGA is able to find higher quality solutions and more consistently locate the optimum compared to a GA. As well as frequently obtaining better quality solutions at the end of the optimisation process, dAGA also finds optimal solutions with fewer evaluations than the GA. On all problems, Table **??** shows that dAGA has a lower number of average evaluations needed to find the optimum, with statistical significance on MAXSAT, 128-HIFF and Knapsack 500.

5.2 Continuous

Results for dAGA, (1+1)-ES and the GA are presented in for the sphere, Table **??** Rosenbrock and Rastrigin functions. On all three of these problems dAGA outperforms the GA, and is able to reach significantly better solutions with fewer evaluations. However, the (1+1)-ES is considerably better on the Sphere and Rosenbrock functions, achieving significantly better solutions than dAGA can over the whole run and in far fewer evaluations. On the Rosenbrock it is almost three times as fast, and on the Sphere can reach the 0.1 target after only 235 evaluations, more than 100 times faster than dAGA. This implies that the (1+1)-ES is a much more efficient continuous optimiser on these search spaces. However, the (1+1)-ES performs extremely badly on the Rastrigin, which consists of many local optima. Here, the population-based GA and dAGA excel, getting much closer to the optimal solution of 0. dAGA is significantly better than the GA and the only algorithm to find solutions under 1.0.

Experiment	Algorithm	Target	Min	Max	Mean	Mean evals
Sphere 50D	GA	0.1	0.065	0.101	0.088 ±0.016	49300.000±4368.829
	(1+1)-ES	0.1	0.000	0.000	0.000 ±0.000	234.900±23.240
	dAGA	0.1	0.026	0.033	0.029 ±0.003	36150.000±502.494
Rosenbrock 10D	GA	0.1	0.547	0.800	0.659 ±0.106	300000.000±0.000
	(1+1)-ES	0.1	0.000	0.000	0.000 ±0.000	27192.200±1186.672
	dAGA	0.1	0.018	0.082	0.042 ±0.018	81950.000±17329.815
Rastrigin 10D	GA	1.0	1.395	4.347	3.058 ±1.234	300000.000±0.000
	(1+1)-ES	1.0	37.808	160.187	89.844 ±33.520	300000.000±0.000
	dAGA	1.0	0.601	1.277	0.888 ±0.203	167800.000±123007.154

Table 2: Results for dAGA, (1+1)-ES and the GA on 3 continuous problems. Showing the value of the minimum and maximum solution returned at the end of search, and the mean best solution in the population after 25%, 50%, 75% and 100% of evaluations had been completed, across 10 trials. A 100,000 evaluation limit is used on sphere and 300,000 is used on the other two. On each function results were found to have significant differences on a one way ANOVA ($p < 0.05$), post-hoc analysis with a t-test and the Bonferroni correction for multiple comparisons showed significant differences between all pairs ($p < 0.05$).

5.3 Accumulation of Adaptations

Results for dAGA on the image generation task described in Section **??** are presented in Fig **??**. This shows the difference in the average number of iterations taken to solve the third problem, when solving the third problem directly and solving it using a dA that has been trained on solving the first two problems without reinitialisation. We see that in every case, there is a significant speed up when dAGA solves the first two problems in the sequence. Fig. 4(a) shows the population created by sampling random inputs after solving the Cross and Box patterns (this set will be referred to as XB), Fig. 4(b) shows the same after solved the Box and Cross patterns (BX) and Fig. 4(c) shows the outputs of the dA generated without training. There is a clear difference in the solutions created, Fig. 4(a) showing box-like patterns mixed with diagonals, Fig. 4(b) showing diagonals mixed with block elements, and Fig. 4(c) showing much noisier solutions. We can see that by training on two solutions, the dA has learnt from both training patterns, and this is exploited to solve the third problem considerably faster, as seen in Fig. **??**.

Unsurprisingly, the largest speed ups occur when the third problem is identical to the second. For both initial training patterns, this speedup is considerably greater than the mean, although it is markedly greater on the BX set. Also notable on BX is that the speed up is much lower on the Box pattern, even though it was already solved in the first run of the algorithm. By training on the Cross pattern second, some information learnt from the first run appears to have been lost. On BX, apart from the Cross, the largest speed up is seen in the solutions which inherit 1/4 from the Box and 3/4 from the Cross. The reverse is not true, and for XB there are large savings on the Cross, which it was trained

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	p
a then p	21.91	33.64	18.82	46.09	10.00	27.82	11.09	84.09	46.27	42.18	43.55	63.45	44.18	53.45	173.64
p then a	117.36	63.18	46.55	65.91	54.45	62.36	44.18	91.55	74.82	61.27	55.91	57.00	72.18	70.36	125.18

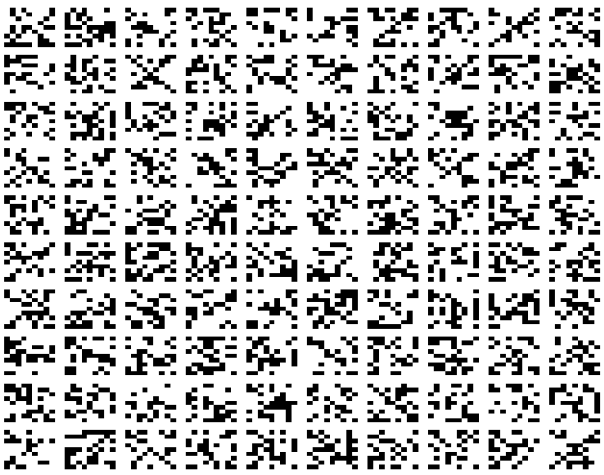
Table 3: Showing the difference between mean number of iterations taken to solve the image labelled on the top row without pre-learning and the mean number of iterations taken with pre learning on the two solutions labelled in the first column, averaged over 10 trials. All mean iterations are significantly different, Wilcoxan Rank Sum test ($p < 0.05$). NOTE: MISSING O!!!!

Fig. 3: Showing the difference between mean number of iterations taken to solve the image pictured on the top row without pre-learning and the mean number of iterations taken with pre learning on the two solutions pictured in the first column, averaged over 10 trials.

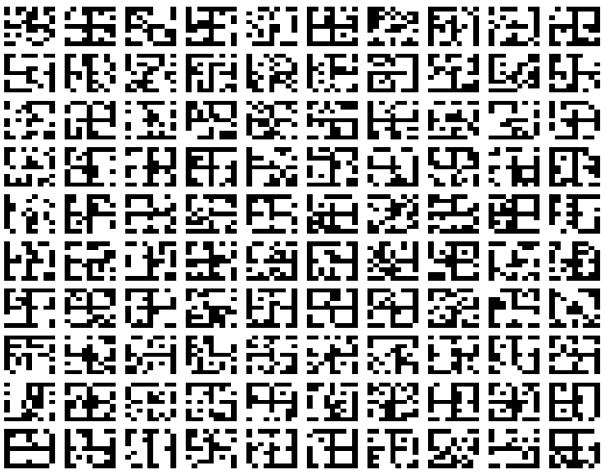
on initially, as well as the 1/4 Box, 3/4 Cross solution that BX performs well on. Fig. 4 (a) and (b) shows that while BX and XB both share features of the two root patterns, they are dominated by the second trained pattern. The Cross patterns are more sparse, which could explain why it takes longer to produce a Box, as more pixels must be filled in. In order to reduce the memory effect, it may help to use a lower learning rate. However, this simple example illustrates that dAGA is able to transfer learned structure between tasks.

6 Discussion

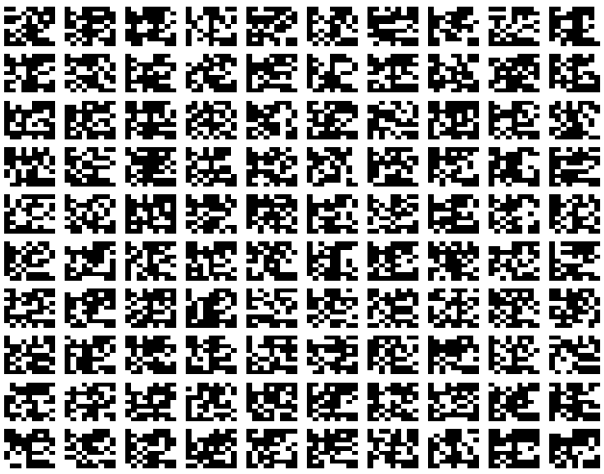
The results from Section 5 show that the dA can be applied to both discrete and continuous problems and regularly outperforms a GA, either in the quality of the final solution or in the number of evaluations used. The advantages over the GA are most clear on difficult combinatorial problems such as the 256-bit HIFF and MAXSAT, where the GA struggles to locate optimal solutions. On these problems, the best dAGA parameters found through a grid search displayed a very low corruption level (0.01 for MAXSAT and 0.05 for 256-HIFF) and large population sizes (2,500 and 5,000). As seen in Fig. 1 the low corruption level maintains variance in the population, maximising the amount of information in the data pool and preventing premature convergence. Conversely, on problems such as the Knapsack and Royal Road, a large corruption level is used. This implies that on problems where there is more independence between input components, a high corruption level can more quickly direct search towards desired regions.



(a)



(b)



(c)

Fig. 4: Showing the output of 100 random inputs passed through the autoencoder

While dAGA outperformed the GA on all three parameter optimisation problems, it was outperformed by a (1+1)-ES on two. There are several improvements that could be made, one of which is the sampling method, which uses a fixed σ^2. Using an adaptive step-size, such as the Evolutionary path length method used in population-based Evolutionary Strategies could have a positive effect on convergence. To solve the continuous problems, dAGA was found to need a large hidden layer, a factor of 2.5 or greater than the input size. Continuous search spaces are much larger than discrete ones and the dA will need a high capacity in order to effectively capture solution structure. This was also found on the 256-bit HIFF, which used 500 hidden neurons. While this could lead to overfitting on the solution set, this can be offset by the corruption applied to the inputs, which acts as a strong regulariser. There is the potential that results could be improved through deeper architectures for the dA, which could capture a hierarchical composition of features, which a shallow architecture cannot. At present we have been unable to successfully apply the algorithm to non-trivial continuous problems with more than 10 dimensions. Further work is needed to determine what architectures can support this added complexity.

The image generation task poses an interesting question that has not been answered by the EDA community - can building a model to solve one task help improve performance on another? The results showed that using a dA, information learnt in one task can be retained and recombined with that learnt in a second task to produce an across the board speed up on the third task in the sequence. This shows the potential for using a dA, but also potentially other modelling techniques from the optimisation community, such as Restricted Boltzmann Machines or Deep Belief Networks, for transfer learning between problems of the same class. Careful experimentation is needed to discover which types of problem can use machine learning techniques to transfer knowledge between unique instances and speed up optimisation as more examples are encountered.

7 Conclusion

We have presented a novel stochastic algorithm based on a denoising autoencoder. Training online, using the best found genotypes, the algorithm is able adaptively learn an exploration distribution, which guides it towards optimal solutions on difficult problems such as the 256-bit HIFF, regularly outperforming a genetic algorithm. In a simple experiment, we have shown that in addition to standard optimisation problems, the dA can learn to generalise across instances in the same problem class, displaying speed ups on unseen instances.

Acknowledgments. The work is funded by the FQEB Templeton grant "Bayes and Darwin", and the FP-7 FET OPEN Grant INSIGHT.

References

1. Pelikan, M., Sastry, K., Cantú-Paz, E.: Scalable optimization via probabilistic modeling. Springer (2006)

2. Pelikan, M., Goldberg, D.E., Lobo, F.G.: A survey of optimization by building and using probabilistic models. Computational optimization and applications **21**(1) (2002) 5–20
3. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. Science **313**(5786) (2006) 504–507
4. Baluja, S.: Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning. Technical report, DTIC Document (1994)
5. Harik, G.: Linkage learning via probabilistic modeling in the ecga. Urbana **51**(61) (1999) 801
6. Harik, G.R., Lobo, F.G., Goldberg, D.E.: The compact genetic algorithm. Evolutionary Computation, IEEE Transactions on **3**(4) (1999) 287–297
7. Martí, L., García, J., Berlanga, A., Molina, J.M.: Introducing moneda: Scalable multiobjective optimization with a neural estimation of distribution algorithm. In: Proceedings of the 10th annual conference on Genetic and evolutionary computation, ACM (2008) 689–696
8. Tang, H., Shim, V.A., Tan, K.C., Chia, J.Y.: Restricted boltzmann machine based algorithm for multi-objective optimization. In: Evolutionary Computation (CEC), 2010 IEEE Congress on, IEEE (2010) 1–8
9. Fernando, C., Goldstein, R., Szathmáry, E.: The neuronal replicator hypothesis. Neural Computation **22**(11) (2010) 2809–2857
10. Zhang, B.T., Shin, S.Y.: Bayesian evolutionary optimization using helmholtz machines. In: Parallel Problem Solving from Nature PPSN VI, Springer (2000) 827–836
11. Pelikan, M., Goldberg, D.E.: Escaping hierarchical traps with competent genetic algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001). (2001) 511–518
12. Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P.A.: Extracting and composing robust features with denoising autoencoders. In: Proceedings of the 25th international conference on Machine learning, ACM (2008) 1096–1103
13. Pelikan, M.: Hierarchical Bayesian optimization algorithm. Springer (2005)
14. Weingartner, H.M., Ness, D.N.: Methods for the solution of the multidimensional 0/1 knapsack problem. Operations Research **15**(1) (1967) 83–103
15. Watson, R.A., Pollack, J.B.: Hierarchically consistent test problems for genetic algorithms. In: Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on. Volume 2., IEEE (1999)
16. Mitchell, M., Forrest, S., Holland, J.H.: The royal road for genetic algorithms: Fitness landscapes and ga performance. In: Proceedings of the first european conference on artificial life, Cambridge: The MIT Press (1992) 245–254
17. Hoos, H., Stiitzle, T.: Satllb: An online resource for research on sat. Sat (2000) 283
18. Watson, R.A., Wagner, G.P., Pavlicev, M., Weinreich, D.M., Mills, R.: The evolution of phenotypic correlations and developmental memory. Evolution (2014)