

Tuning Differential Evolution For Artificial Neural Networks

By

Magnus Erik Hvass Pedersen, Andrew John Chipperfield
Hvass Laboratories Technical Report no. HL0803, 2008

Abstract

The efficacy of an optimization method often depends on the choosing of a number of behavioural parameters. Research within this area has been focused on devising schemes for adapting the behavioural parameters during optimization, so as to alleviate the need for a practitioner to select the parameters manually. But these schemes usually introduce new behavioural parameters that must be tuned. This study takes a different approach in which finding behavioural parameters that yield good performance is considered an optimization problem in its own right and can therefore be attempted solved by an overlaid optimization method. In this work, variants of the general purpose optimization method known as Differential Evolution have their behavioural parameters tuned so as to work well in the optimization of an Artificial Neural Network. The results show that DE variants using so-called adaptive parameters do not have a general performance advantage as previously believed.

Keywords: Numerical optimization, stochastic, multi-agent, parameter tuning.

1 Introduction

The optimization method known as Differential Evolution (DE) was originally introduced by Storn and Price [1] and offers a way of optimizing a given problem without explicit knowledge of the gradient of that problem. This is particularly useful if the gradient is difficult or even impossible to derive. Gradient-based optimization for the dataflow model known as an Artificial Neural Network (ANN) is called BackPropagation (BP) and was developed independently by Werbos and Rumelhart et al. [2] [3] [4]. Deriving the gradient for an optimization problem is not always trivial, so optimizers such as DE which do not rely on the gradient can be used in rapid prototyping of new ANN variants and mathematical models in general.

Optimizing an ANN is an interesting problem because it may easily contain a large number of parameters that are highly interrelated and do not lend themselves well to direct human interpretation. DE has been used for optimizing ANNs by Ilonen et al. [5] who conclude that DE has performance comparable to the classic gradient-based approach, although results presented show that the gradient-based BP consistently out-performs DE. It has been recognized since the inception of DE that different choices of behavioural parameters cause it to perform worse or better on particular problems and the selection of good parameters is a challenging art, see for example Storn et al. [6] [7], Liu and Lampinen

[8], and Zaharie [9]. There has been a trend in recent years to try and make the DE parameters automatically adapt to new problems during optimization, hence alleviating the need for the practitioner to select the parameters by hand, see for example Price et al. [7], Liu and Lampinen [10], Qin et al. [11] [12], and Brest et al. [13]. But these DE variants with so-called adaptive parameters just introduce new parameters that must then be tuned by the practitioner and has therefore merely deferred this difficult issue without actually eliminating it.

Finding the best choice of behavioural parameters for an optimization method is considered here to be an optimization problem in its own right and hence attempted solved by an overlaying optimization method. This is known as Meta-Optimization, Meta-Evolution, Super-Optimization, etc. See for example the work by Grefenstette [14], Bäck [15], Keane [16], Meissner et al. [17], and the more statistically oriented approaches by François and Lavergne [18], Czarn et al. [19], and Nannen and Eiben [20]. We have previously used a meta-optimization technique that is both simple and efficient for tuning DE parameters to perform well on benchmark problems [21], and that technique is also used here for tuning the DE parameters to perform well in ANN optimization, which poses other challenges.

The paper is organized as follows. Section 2 gives a brief overview of the class of ANNs to be optimized. Section 3 describes the DE variants considered in this work. Section 4 describes an optimization method especially suited for meta-optimization, and section 5 describes how to employ it as an overlaid meta-optimizer. The experimental settings and results are given in section 6, and overall conclusions are found in section 7.

2 Artificial Neural Networks

An ANN is a dataflow model inspired by the complex connections of cells in a biological brain. The type of ANN used in this study is a fully connected feedforward network with a single hidden layer having four nodes. This means the data flows from an input layer, through a single hidden layer, and finally reaches the output layer. This kind of ANN has its origin in research dating back several decades (see [22, Section 1.9] for a detailed historical account). More recent texts on this and other kinds of ANN can be found in [22] [23]. The ANN has parameters, usually called weights, that can be adjusted to provide different mappings from input to output. The optimization task is then to change these weights so a desired correspondence between inputs and outputs of the ANN is achieved. Under certain circumstances, and provided the input-output mapping is well-behaved, this approximation can be achieved arbitrarily well [24] [25] [26], simply by adding more hidden nodes to an ANN having a single hidden layer whose nodes use bounded, compressive functions such as the Sigmoid function described below. However, the datasets used in the experiments here are not necessarily well-behaved as they may contain contradictory entries or have missing features needed to properly distinguish them.

To briefly describe the computation of this kind of ANN, let the j 'th node of

the i 'th layer be denoted by n_{ij} . The node has a bias-value b_{ij} for offsetting the influence of its inputs. The weight denoted by w_{ijk} connects the k 'th node from the previous layer to node n_{ij} . For an ANN having just a single hidden layer, the computation of the j 'th element of the ANN output \vec{o} can be expressed as a single, flattened formula:

$$o_j = b_{3,j} + \sum_{k=1}^{N_2} w_{3,j,k} \cdot \sigma \left(b_{2,k} + \sum_{l=1}^{N_1} w_{2,k,l} \cdot n_{1,l} \right) \quad (1)$$

Where $\vec{n}_1 \in \mathbb{R}^{N_1}$ is the first layer of the network containing its input, and N_1 is the number of nodes in this first layer. The number of nodes in the hidden layer is N_2 , and the number of nodes in the output layer is N_3 . The parameters subject to optimization are all the bias-values and weights for the entire ANN, that is, b_{ij} and w_{ijk} for all valid combinations of i , j , and k . The $\sigma(\cdot)$ part of this formula represents the calculation of the k 'th node in the hidden layer of the ANN, where the compressive function used here is called the Sigmoidal function $\sigma : \mathbb{R} \rightarrow (0, 1)$, defined as: $\sigma(x) = 1/(1 + e^{-x})$.

A fitness measure must first be defined in order to optimize the weights and bias-values of an ANN. Let T be the set of input-output data the ANN must be trained to mimic. An element in the dataset T consists of input data \vec{t}_i and its associated output data \vec{t}_o . Passing the input data \vec{t}_i to the ANN and computing the actual ANN output using Eq.(1) yields \vec{o} . Accumulating the error between desired and actual ANN output for all data pairs in T gives a measure for how well the ANN mimics a desired input-output mapping. This is known as the Mean Squared Error (MSE), defined as:

$$\text{MSE} = \frac{1}{N_3 \cdot |T|} \sum_{(\vec{t}_i, \vec{t}_o) \in T} \sum_{j=1}^{N_3} (o_j - t_{oj})^2 \quad (2)$$

where normalization is done with both the number of input-output pairs $|T|$, and the number of ANN output-nodes N_3 . This is not standard in the literature, but has the advantage of giving a uniform fitness measure that can be compared independently of the given dataset and ANN sizes. The MSE measure is also sometimes halved in the literature to facilitate a cleaner derivation of the gradient, but that is ignored here. It is customary to split the dataset T and use one part during optimization (or training) of the ANN, and the other part to test the generalization ability of the ANN, as a simple form of statistical cross-validation [27]. However, this is ignored here as this study is primarily concerned with the performance of DE rather than that of the ANN.

The traditional way of optimizing ANN weights is to use a gradient-based optimization method to follow the path of steepest descent of the MSE measure. This is known as BackPropagation (BP) because it first makes a forward pass and computes the node values for the entire ANN, and then propagates the MSE errors backwards through the ANN to determine adjustments for the weights and bias-values. The BP method was developed independently by Werbos and

Rumelhart et al. [2] [3] [4]. The stochastic gradient is used in this study, where a single input-output pair is picked randomly from the dataset T , and the gradient is computed for just that data pair. This requires much less computational time than accumulating the gradient for all data pairs in T , and taking small enough steps yields satisfactory results [28].

Once the ANN weights and bias-values have been optimized for the MSE measure, the ANN can be used for classification tasks as well, as proven by Richard and Lippmann [29]. This study uses 1-of- m classification [22] [28] [30], where each ANN output represents a class, and the output with the highest value is chosen as the classification. The Classification Error (CLS) is computed as the number of data-pairs in T , for which the ANN output classification does not match that of the desired output, divided by the total number of data pairs $|T|$. This yields a CLS value in the range $[0, 1]$, where a lower CLS value means a better classification rate.

3 Differential Evolution

An optimization method that can be used instead of BP for optimizing ANN weights and bias-values, is the population-based method known as Differential Evolution (DE) due to Storn and Price [1]. DE does not use the gradient of the problem it is optimizing, but maintains a number of candidate solutions, here called agents, creating new agents by combining randomly chosen agents from its population, and accepting the new agents in case of fitness improvement.

3.1 Basic Variants

DE employs evolutionary operators that are dubbed crossover and mutation in its attempt to minimize some fitness function $f : \mathbb{R}^n \rightarrow \mathbb{R}$. The operators are typically applied in turn but have been combined for a more concise description in the following. The basic DE/rand/1/bin variant is believed to be one of the best performing and hence one of the most popular of the basic DE variants [7] [31]. Formally, let $\vec{y} \in \mathbb{R}^n$ be the new candidate solution for an agent whose current position in the search-space is \vec{x} , and let \vec{a} , \vec{b} , and \vec{c} be the positions of distinct and randomly chosen agents, which must also be distinct from the agent \vec{x} that is currently being updated. The elements of $\vec{y} = [y_1, \dots, y_n]$ are then computed as follows:

$$y_i = \begin{cases} a_i + F \cdot (b_i - c_i) & , i = R \vee r_i < CR \\ x_i & , \text{else} \end{cases} \quad (3)$$

where $F \in [0, 2]$ is a user-adjustable parameter called the differential weight, and the randomly chosen index $R \in \{1, \dots, n\}$ ensures at least one element differs from that of the original agent: $y_R \neq x_R$. While the rest of the elements are either chosen from the original position \vec{x} or computed from combining other agents' positions, according to the user-adjustable crossover probability $CR \in [0, 1]$, and with $r_i \sim U(0, 1)$ being a uniform random number drawn for

each vector element i . Once the new potential position \vec{g} has been computed, it will replace the agent's current position \vec{x} in the case of improvement to the fitness. An additional user-adjustable parameter of DE is the population size NP , that is, the number of agents in the DE population.

The DE/best/1/bin variant has been long out of favour with researchers and practitioners because it is believed to have inferior performance with tendencies for premature convergence [7] [31]. The DE/best/1/bin variant replaces Eq.(3) with:

$$y_i = \begin{cases} g_i + F \cdot (a_i - b_i) & , i = R \vee r_i < CR \\ x_i & , \text{else} \end{cases} \quad (4)$$

where \vec{g} is the population's best known position until now. In the original version of this, the agents \vec{a} and \vec{b} are chosen to be different not only from each other, but also from the agent \vec{x} currently being processed, but it simplifies the implementation a good deal if the latter is not required. This variant is called DE/best/1/bin/simple, nicknamed The Joker, and was shown in [21] to have an overall good performance

if the behavioural parameters were properly tuned.

3.2 Perturbed & Adaptive Parameters

An attempt to remedy the need for a user to determine DE parameters that are good for a given optimization problem, is to perturb the behavioural parameters during optimization. Several schemes for perturbing the differential weight F have been proposed in the literature [7] [32], where the ones used by Storn himself are generally just the Dither and Jitter schemes [31] [33]. It was noted in [21] that these two schemes are actually identical, with the exception of Dither perturbing F once for every vector being processed in Eq.(3) or (4), where as Jitter perturbs it for each vector-element being processed. A common notation may therefore be used for both Dither and Jitter, having a midpoint F_{mid} and a range F_{range} for the perturbation:

$$F \sim U(F_{mid} - F_{range}, F_{mid} + F_{range})$$

The midpoint can be anywhere between $F_{mid} \in [0, 2]$, and the perturbation range can be anywhere between $F_{range} \in [0, 3]$, which were chosen to allow for unusual values when F_{mid} and F_{range} are tuned automatically later in this study.

The concept of perturbing behavioural parameters can be taken a step further, by also using the fitness improvement to guide selection of behavioural parameters that yielded good performance. Examples of such adaptive schemes for DE parameters are the Fuzzy Adaptive DE (FADE) by Liu and Lampinen [10], and the Self-adaptive DE (SaDE) due to Qin et al. [11] [12]. In this study the DE variant known as Janez DE (JDE) will be used, which is due to Brest et al. [13], and has been chosen because its performance has been found to compare well against other so-called state-of-the-art DE variants [13] [34]. It is presented by its authors as a Self-Adaptive DE as well, because it eliminates the

need for a user to select the F and CR parameters - yet ironically introduces 8 new user-adjustable parameters to achieve this. This tendency is common for such DE variants, and indeed also for the comparatively simpler Dither and Jitter variants described above which merely perturb the behavioural parameters. The JDE variant works as follows. First assign start values to F and CR , F_{init} and CR_{init} respectively. Then before computing the new potential position of a DE agent using Eq.(3) or (4), decide what parameters F and CR to use in that formula. With probability $\tau_F \in [0, 1]$ draw a new random $F \sim U(F_l, F_l + F_u)$, otherwise reuse F from previously, where each DE agent retains its own F parameter. Similarly each agent retains its own CR parameter, for which a new random value $CR \sim U(CR_l, CR_l + CR_u)$ is picked with probability $\tau_{CR} \in [0, 1]$ otherwise the old CR value for that agent is reused. Whichever F and CR values are being used in the computation of Eq.(3) or (4), they will survive to the next iteration or be discarded along with the agent’s new potential position \vec{y} , according to fitness improvement.

4 Local Unimodal Sampling

To tune the behavioural parameters of DE variants an overlaid optimization method is used which is called Local Unimodal Sampling (LUS) and is described in more detail in [21]. Briefly, LUS is somewhat related to the Matyas [35] and Rastrigin [36] families of optimization methods, which work by sampling a fraction of the search-space surrounding the current position, and moving to the new position in case of improvement to the fitness. LUS samples the new potential position $\vec{y} \in \mathbb{R}^n$ from the neighbourhood of the current position \vec{x} by adding a random vector \vec{a} :

$$\vec{y} = \vec{x} + \vec{a}, \vec{a} \sim U(-\vec{d}, \vec{d})$$

with \vec{d} being the current search-range, initially chosen as the full range of the search-space and decreased during an optimization run as described below. When LUS is used for meta-optimization these represent different choices of DE behavioural parameters, meaning e.g. that $\vec{x} = [NP, CR, F]$ for DE/rand/1/bin. The search-space therefore constitutes all valid choices of behavioural parameters, whose boundaries will be detailed later.

LUS decreases its sampling-range during optimization so as to focus the search, in a manner related to those described in earlier methods [37] [38] [39] [40] [41] [42], only more simply. When a sample fails to improve the fitness, the search-range \vec{d} is decreased by multiplying with a factor q , defined as:

$$\vec{d} \leftarrow q \cdot \vec{d}, q = 2^{-\beta/n} \tag{5}$$

where n is the dimensionality of the search-space, and $0 < \beta < 1$ causes slower decrease of the search-range, where as $\beta > 1$ causes more rapid decrease. For the experiments in this paper, a value of $\beta = 1/3$ is used as it has been found to yield good results on a broad range of problems.

5 Meta-Optimization

The parameters controlling the behaviour and efficacy of DE are usually found by manual experimentation, which is time-consuming and susceptible to human misconceptions of the inner-working of the optimization method. The problem of finding the best choice of behavioural parameters for a given optimization method, is considered here as an optimization problem in its own right and is termed Meta-Optimization. In other words, the idea is to have an optimization method act as an overlaying meta-optimizer, trying to find the best performing behavioural parameters for another optimization method, which in turn is used to optimize the ANN weights. The overall concept is depicted in figure 1. It is important to understand that parameter tuning by way of meta-optimization is done in an offline manner, and hence differs from adaptation of DE parameters which is done in an online manner during optimization. The DE parameters found using meta-optimization can therefore be used by other researchers and practitioners without the need for augmenting their DE implementation with adaptive schemes. This also means that meta-optimization can be readily employed on other optimization methods than just DE. The approach to meta-optimization from [21] is used here, albeit with a simplification because the tuning is done with regard to just one ANN problem, where as [21] would tune the DE parameters for multiple problems.

The crux of automatically finding good behavioural parameters for an optimization method, is to define an appropriate meta-fitness measure that can be made the subject of meta-optimization. The meta-fitness measure must reflect how the optimization method is ultimately to be used, but at the same time allow for efficient meta-optimization. A typical way of performing optimization with DE is to let it run for some predetermined and seemingly adequate number of iterations. This means the performance of DE and a given choice of behavioural parameters, can be rated in terms of the ANN fitness that can be obtained within this number of iterations. Since DE is stochastic by nature, it will be likely that it gives a different result for each optimization run, and a simple way of lessening this stochastic noise is to perform a number of optimization runs in succession, using the average of the fitness values obtained to guide the meta-optimization. But repeating optimization runs of DE also causes the computational time to grow undesirably. A simple way of saving computational time is to preemptively abort a meta-fitness evaluation, once the meta-fitness becomes worse than that needed for the overlaying meta-optimizer to accept the new parameters, and the meta-fitness is known not to improve for the rest of the evaluation. This technique is generally termed Preemptive Fitness Evaluation and has been used by researchers for decades, although its original author is difficult to establish as the technique is seldom mentioned in the literature.

To employ preemptive fitness evaluation when meta-optimizing the DE parameters, the meta-fitness measure must be ensured to be non-decreasing and hence only able to grow worse, so as the evaluation can be aborted safely without risk of it improving in later parts of its evaluation. Since the global minimum for the underlying ANN fitness measure is zero (see Eq.(2)), the best perfor-

mance of DE when optimizing ANN weights is also a meta-fitness value of zero. The algorithm for computing the meta-fitness is shown in figure 2, where the preemptive fitness limit is denoted L , and is the limit beyond which the meta-fitness evaluation can be aborted. This limit is passed as an argument by the overlaying LUS meta-optimizer, so that L is the meta-fitness of the currently best known choice of DE parameters. Depending on the experimental settings, the time-saving resulting from the use of preemptive fitness evaluation in meta-optimization, ranges from approximately 50% to 85%.

6 Experimental Results

This section gives the experimental settings and results for optimizing ANN weights using DE variants with their behavioural parameters tuned accordingly.

6.1 Optimization Settings

Five ANN datasets are used in this study and they are all taken from the Proben1 library [30]. These particular datasets are chosen because they give acceptable MSE results within a reasonable number of optimization iterations. All five datasets are classification problems and their specifications are detailed in table 1. The Cancer dataset diagnoses a breast tumour as being either benign or malignant from various microscopic measurements. The Card dataset determines whether a credit-card can be granted an applicant by a financial institute according to various data of that applicant. The Gene dataset detects intron/exon boundaries in genetic sequences from a window of 60 DNA sequence elements. The Soybean dataset recognizes diseases in soybeans according to various measurements on the soybean and the plant itself. The Thyroid dataset diagnoses performance of the thyroid from patient query data.

To lessen the effect of stochastic variation 50 optimization runs are performed on each ANN problem and the results averaged. In each of these ANN optimization runs a number of iterations are executed which equals 20 times the total number of weights and bias-values for the problem in question. For instance, as the ANN Gene problem can be seen from table 1 to have 499 weights, it means 9980 optimization iterations are performed in each run. An iteration is here taken to mean a single fitness evaluation of the MSE measure. Using these settings the ANN Cancer problem is the fastest to optimize and takes approximately 25 seconds for all 50 runs, while the ANN Gene problem takes approximately 80 minutes for all 50 optimization runs, when executed on an Intel Pentium M 1.5 GHz laptop computer using an implementation in the ANSI C programming language.

All ANN weights and bias-values are initially picked randomly and uniformly from the range $(-0.05, 0.05)$, which works well for classic BP and DE alike:

$$\forall i, j, k : w_{ijk} \sim U(-0.05, 0.05), b_{ij} \sim U(-0.05, 0.05)$$

Boundaries for the weights and bias-values are used to limit the search for optima. These boundaries were chosen from experiments with several different optimization methods, suggesting they are applicable in general. Although classic BP does not require such boundaries, the values chosen do not impair the performance for the class of problems considered here. The boundaries are:

$$\forall i, j, k : w_{ijk} \in [-7, 7], b_{ij} \in [-7, 7]$$

When an optimizing agent steps outside the boundaries of the search-space, it will be moved back to the boundary value.

6.2 Meta-Optimization Settings & Results

To fairly compare the performance of DE variants against each other, their behavioural parameters will have to be tuned to perform their best. It is recommended in [21] that experiments be done to uncover the specialization and generalization ability of an optimizer. Here, however, these experiments are combined so the DE parameters will be meta-optimized to perform well on a single ANN problem and those parameters will be used on the remainder of the ANN problems as well.

The experimental settings for meta-optimization are as follows. Six meta-optimization runs are conducted with the LUS method as the overlaying meta-optimizer trying to find the best performing behavioural parameters of DE. Each meta-optimization run has a number of iterations equalling 20 times the number of parameters to be optimized. So for basic DE which has three behavioural parameters, namely NP , CR , and F , 60 iterations will be performed for each meta-optimization run of the overlaying LUS method. DE is in turn made to perform 10 optimization runs on the ANN Cancer problem in each of these iterations, as described above. The Cancer problem is used for meta-optimization because it is the one that is fastest to compute. These meta-optimization settings were found to be adequate in locating the behavioural parameters that cause the DE variants to perform well on that ANN problem. Meta-optimization using these settings requires approximately 30 minutes of computation time per DE variant.

The boundaries for the DE parameter search-spaces are shown in table 2, where it should be noted that we allow for smaller population sizes NP than usually [6] [7] [8], in part because so few optimization iterations are allowed here and each DE agent will therefore perform more iterations, but also because these have been found to be adequate, and the smaller a parameter search-space is the faster is the discovery of good parameter choices using meta-optimization.

Table 3 shows the best performing parameters for the DE/rand/1/bin variants found through meta-optimization. These parameters have abnormally small population sizes NP , compared to the advise typically given in the literature [1] [6] [8] [32], although small populations have been reported to work well on certain problems [7]. The differential weight F is also somewhat different from commonly advised, especially concerning the amount of perturbation in the

Dither and Jitter variants, e.g. Jitter is recommend used with $F_{range} = 0.0005$ [32]. However, the crossover probability CR is similar to that often recommended. The JDE parameters seem to follow the same tendencies as the simpler DE variants, although their intrinsic meaning is more difficult to establish due to the increased complexity of that DE variant. The meta-optimized parameters for the DE/best/1/bin/simple variants are also shown in table 3. These parameters differ from the DE/rand/1/bin parameters in that they use larger population sizes NP , higher crossover probabilities CR , and smaller differential weights F . Deducing this from a conceptual or philosophical study of how these DE variants actually work would seem to be impossible, whereas meta-optimization allows us to automatically find good choices of behavioural parameters, without knowing the analytic cause of their good performance.

6.3 Results of ANN Weight Optimization

Table 4 shows the results of using these DE variants to optimize the weights for all the ANN problems, not just the Cancer problem for which the DE parameters were specifically tuned. The results show that the DE/best/1/bin/simple variants generally have an advantage over the DE/rand/1/bin variants, but there does not appear to be a general advantage to using adaptive schemes over fixed DE parameters, as long as they have been tuned properly.

Figures 3-7 show the averaged fitness traces for the optimization runs that led to the results in table 4. What is important to note from these fitness traces, is that they are not only similar in terms of the end results, but also in terms of the rate of convergence. This is remarkable because it indicates the underlying dynamic behaviour of these DE variants are indeed similar.

Comparing the DE results to those of using classic BP in table 5 (using a hand-tuned step-size of 0.05 common in the literature, see e.g. [28]), it can be seen that DE is not quite capable of the same general performance as BP, but DE performs well enough to be a viable optimizer, for instance in the development of new kinds of ANN for which gradients have not yet been derived.

7 Conclusion

Experiments were conducted with meta-optimizing the behavioural parameters of eight different DE variants when used for optimizing ANN weights. The DE variants ranged from the basic having fixed behavioural parameters during optimization, to DE variants employing some form of perturbation or adaptation of the parameters, designed with the intent of better

adapting and generalizing to new optimization problems. The results show that there is no general advantage to these so-called adaptive parameter schemes, when compared to the basic DE which has fixed behavioural parameters during optimization, if only those parameters are chosen properly. We suggest using an automated process to parameter tuning such as the meta-optimization technique presented, which is both simple and effective, and does not require for

the practitioner to attain special knowledge about what causes good performance for the optimizer in question, as coarse parameter ranges will suffice for initializing the meta-optimization process.

8 Acknowledgements

Dr. Rainer Storn (researcher at Rohde & Schwarz, Germany) is thanked for suggestions on what DE variants to be used in this study. Professor Emeritus Rein Luus (University of Toronto, Canada) is thanked for his comments to the first draft of this paper. Associate Professor Janez Brest (University of Maribor, Slovenia) is thanked for supplying source-code for the JDE variant.

9 Source-Code

Source-code implemented in the ANSI C programming language and used in the experiments in this paper, can be found in the SwarmOps library on the internet address: <http://www.Hvass-Labs.org/>

References

- [1] R. Storn and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341 – 359, 1997.
- [2] P.J. Werbos. *Beyond Regression: new tools for prediction and analysis in the behavioural sciences*. PhD thesis, Harvard University, Cambridge, MA, 1974.
- [3] P.J. Werbos. *The Roots of Backpropagation: from ordered derivatives to neural networks and political forecasting*. Wiley-Interscience, 1994.
- [4] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1: foundations*, pages 318–362. MIT Press, 1986.
- [5] J. Ilonen, J-K. Kamarainen, and J. Lampinen. Differential evolution training algorithm for feed-forward neural networks. *Neural Processing Letters*, 17(1):93–105, 2003.
- [6] R. Storn. On the usage of differential evolution for function optimization. In *Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS)*, pages 519–523, Berkeley, CA, USA, 1996.
- [7] K. Price, R. Storn, and J. Lampinen. *Differential Evolution – A Practical Approach to Global Optimization*. Springer, 2005.

- [8] J. Liu and J. Lampinen. On setting the control parameter of the differential evolution method. In *Proceedings of the 8th International Conference on Soft Computing (MENDEL)*, pages 11–18, Brno, Czech Republic, 2002.
- [9] D. Zaharie. Critical values for the control parameters of differential evolution algorithms. In *Proceedings of MENDEL 2002, 8th International Mendel Conference on Soft Computing*, pages 62–67, Bruno, 2002.
- [10] J. Liu and J. Lampinen. A fuzzy adaptive differential evolution algorithm. *Soft Computing*, 9(6):448–462, 2005.
- [11] A.K. Qin and P.N. Suganthan. Self-adaptive differential evolution algorithm for numerical optimization. In *Proceedings of the IEEE congress on evolutionary computation (CEC)*, pages 1785–1791, 2005.
- [12] A.K. Qin, V.L. Huang, and P.N. Suganthan. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions on Evolutionary Computation, Accepted*, 2008.
- [13] J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Žumer. Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark functions. *IEEE Transactions on Evolutionary Computation*, 10(6):646–657, 2006.
- [14] J.J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions Systems, Man, and Cybernetics*, 16(1):122–128, 1986.
- [15] T. Bäck. Parallel optimization of evolutionary algorithms. In *Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature (PPSN)*, pages 418–427, London, UK, 1994. Springer-Verlag.
- [16] A.J. Keane. Genetic algorithm optimization in multi-peak problems: studies in convergence and robustness. *Artificial Intelligence in Engineering*, 9:75–83, 1995.
- [17] M. Meissner, M. Schmuker, and G. Schneider. Optimized particle swarm optimization (OPSO) and its application to artificial neural network training. *BMC Bioinformatics*, 7(125), 2006.
- [18] O. François and C. Lavergne. Design of evolutionary algorithms – a statistical perspective. *IEEE Transactions on Evolutionary Computation*, 5(2):129–148, 2001.
- [19] A. Czarn, C. MacNish, K. Vijayan, B. Turlach, and R. Gupta. Statistical exploratory analysis of genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 8(4):405–421, 2004.

- [20] V. Nannen and A.E. Eiben. A method for parameter calibration and relevance estimation in evolutionary algorithms. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO)*, pages 183–190, Seattle, USA, 2006.
- [21] M.E.H. Pedersen and A.J. Chipperfield. Parameter tuning versus adaptation: proof of principle study on differential evolution. Technical Report HL0802, Hvass Laboratories, 2008.
- [22] S. Haykin. *Neural Networks: a comprehensive foundation*. Prentice Hall, 2nd edition, 1999.
- [23] C.M. Bishop. *Neural Networks For Pattern Recognition*. Oxford University Press, 1995.
- [24] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.
- [25] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.
- [26] A.R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, 1993.
- [27] L.K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.
- [28] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
- [29] M.D. Richard and R.P. Lippmann. Neural network classifiers estimate bayesian a-posteriori probabilities. *Neural Computation*, 3(4):461–483, 1991.
- [30] L. Prechelt. Proben1 – a set of neural network benchmark problems and benchmarking rules. Technical Report 21/94, Faculty of Informatics, University of Karlsruhe, Germany, 1994.
- [31] R. Storn. Private correspondance, 2008.
- [32] R. Storn. Differential evolution research – trends and open questions. In U. K. Chakraborty, editor, *Advances in Differential Evolution*, chapter 1. Springer, 2008.
- [33] R. Storn. Optimization of wireless communications applications using differential evolution. In *SDR Technical Conference*, Denver, 2007.
- [34] J. Brest, B. Bošković, S. Greiner, V. Žumer, and M.S. Maučec. Performance comparison of self-adaptive and adaptive differential evolution algorithms. *Soft Computing*, 11:617–629, 2007.

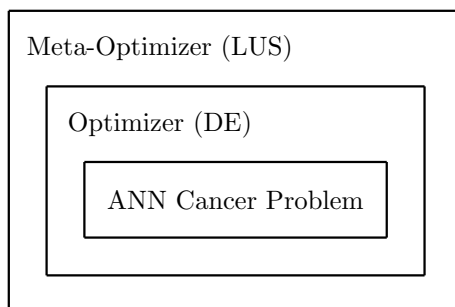


Figure 1: The concept of meta-optimization. The LUS optimization method is used as an overlaid meta-optimizer for finding good behavioural parameters of DE, which in turn is used to optimize the ANN weights for the Cancer problem.

- [35] J. Matyas. Random optimization. *Automation and Remote Control*, 26(2):246–253, 1965.
- [36] L.A. Rastrigin. The convergence of the random search method in the extremal control of a many parameter system. *Automation and Remote Control*, 24(10):1337–1342, 1963.
- [37] V.A. Mutseniyeks and L.A. Rastrigin. Extremal control of continuous multi-parameter systems by the method of random search. *Engineering Cybernetics*, 1:82–90, 1964.
- [38] M.A. Schumer and K. Steiglitz. Adaptive step size random search. *IEEE Transactions on Automatic Control*, 13(3):270–276, 1968.
- [39] J.P. Lawrence III and K. Steiglitz. Randomized pattern search. *IEEE Transactions on Computers*, C-21(4):382–385, 1972.
- [40] R. Luus and T.H.I. Jaakola. Optimization by direct search and systematic reduction of the size of search region. *American Institute of Chemical Engineers Journal (AIChE)*, 19(4):760–766, 1973.
- [41] G. Schrack and M. Choit. Optimized relative step size random searches. *Mathematical Programming*, 10(1):230–244, 1976.
- [42] F.J. Solis and R.J-B. Wets. Minimization by random search techniques. *Mathematics of Operation Research*, 6(1):19–30, 1981.

-
- Initialize the run-counter: $i \leftarrow 1$, and the fitness-sum: $s \leftarrow 0$.
 - While ($i \leq M$) and ($s < L$), do:
 - Perform an optimization run on an ANN problem using DE with the given choice of behavioural parameters.
 - Add the best fitness obtained in the run (call it \bar{f}) to the fitness-sum: $s \leftarrow s + \bar{f}$.
 - Increment the run-counter: $i \leftarrow i + 1$.
 - Return s to the overlaying LUS meta-optimizer as the meta-fitness value of DE with the given choice of behavioural parameters.
-

Figure 2: Algorithm for performing a single meta-fitness evaluation, for rating the performance of DE using a given choice of behavioural parameters.

Problem	Inputs	Classes	Weights	Data Pairs
Cancer	9	2	50	699
Card	51	2	218	690
Gene	120	3	499	3175
Soybean	35	19	427	683
Thyroid	21	3	103	7200

Table 1: ANN dataset specifications, giving for each dataset the number of inputs, the number of possible classifications, the total number of ANN weights and bias-values when using an ANN with a single hidden layer having 4 nodes, and the number of input-output pairs in the dataset.

DE Variant	Behavioural Parameters		
Basic	$NP \in \{4, \dots, 200\}$	$CR \in [0, 1]$	$F \in [0, 2]$
Dither	$NP \in \{4, \dots, 200\}$	$CR \in [0, 1]$	$F_{mid} \in [0, 2]$ $F_{range} \in [0, 3]$
Jitter	$NP \in \{4, \dots, 200\}$	$CR \in [0, 1]$	$F_{mid} \in [0, 2]$ $F_{range} \in [0, 3]$
JDE	$NP \in \{4, \dots, 200\}$	$CR_{init} \in [0, 1]$ $CR_l \in [0, 1]$ $CR_u \in [0, 1]$ $\tau_{CR} \in [0, 1]$	$F_{init} \in [0, 2]$ $F_l \in [0, 2]$ $F_u \in [0, 2]$ $\tau_F \in [0, 1]$

Table 2: Boundaries for the parameter search-spaces of DE variants as used in the meta-optimization experiments.

Variant		Behavioural Parameters		
DE/rand/1/bin	Basic	$NP = 9$	$CR = 0.804681$	$F = 0.736314$
	Dither	$NP = 9$	$CR = 0.824410$	$F_{mid} = 0.833241$ $F_{range} = 1.553993$
	Jitter	$NP = 7$	$CR = 0.916751$	$F_{mid} = 0.809399$ $F_{range} = 0.716685$
	JDE	$NP = 9$	$CR_{init} = 0.658907$ $CR_l = 0.835000$ $CR_u = 0.034128$ $\tau_{CR} = 0.774923$	$F_{init} = 0.946048$ $F_l = 0.400038$ $F_u = 0.807296$ $\tau_F = 0.242418$
DE/best/1/bin/simple	Basic	$NP = 44$	$CR = 0.967665$	$F = 0.536893$
	Dither	$NP = 47$	$CR = 0.954343$	$F_{mid} = 0.391178$ $F_{range} = 0.843602$
	Jitter	$NP = 13$	$CR = 0.933635$	$F_{mid} = 0.513652$ $F_{range} = 1.007801$
	JDE	$NP = 18$	$CR_{init} = 0.777215$ $CR_l = 0.889368$ $CR_u = 0.110632$ $\tau_{CR} = 0.846782$	$F_{init} = 1.393273$ $F_l = 0.319121$ $F_u = 0.933712$ $\tau_F = 0.619482$

Table 3: DE parameters tuned for the ANN Cancer problem.

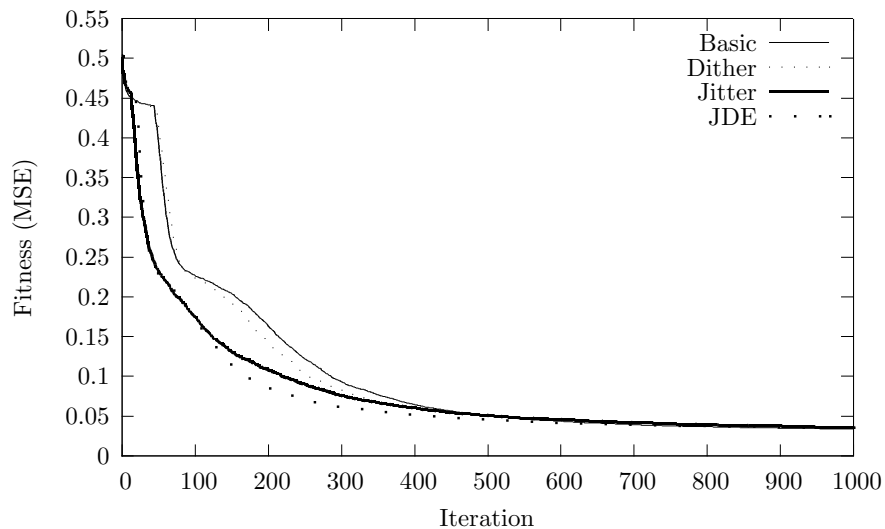


Figure 3: Fitness traces from using DE/best/1/bin/simple variants to optimize the ANN **Cancer** problem. Plot shows the average fitness obtained at each iteration of 50 optimization runs.

	Variant	DE/rand/1/bin		DE/best/1/bin/simple	
		MSE	CLS	MSE	CLS
Cancer	Basic	0.043 (0.011)	0.037 (0.007)	0.034 (0.004)	0.033 (0.004)
	Dither	0.046 (0.010)	0.039 (0.008)	0.035 (0.004)	0.033 (0.004)
	Jitter	0.046 (0.013)	0.040 (0.009)	0.036 (0.007)	0.032 (0.005)
	JDE	0.043 (0.009)	0.039 (0.007)	0.035 (0.004)	0.033 (0.004)
Card	Basic	0.097 (0.008)	0.123 (0.010)	0.091 (0.004)	0.118 (0.007)
	Dither	0.114 (0.017)	0.135 (0.023)	0.091 (0.004)	0.119 (0.007)
	Jitter	0.099 (0.011)	0.124 (0.017)	0.094 (0.011)	0.117 (0.012)
	JDE	0.105 (0.010)	0.127 (0.009)	0.098 (0.008)	0.122 (0.006)
Gene	Basic	0.143 (0.010)	0.288 (0.038)	0.125 (0.009)	0.236 (0.037)
	Dither	0.158 (0.010)	0.331 (0.039)	0.122 (0.011)	0.222 (0.039)
	Jitter	0.154 (0.009)	0.322 (0.037)	0.138 (0.011)	0.273 (0.051)
	JDE	0.155 (0.009)	0.334 (0.038)	0.139 (0.010)	0.281 (0.042)
Soybean	Basic	0.041 (0.001)	0.588 (0.054)	0.038 (0.001)	0.580 (0.052)
	Dither	0.042 (0.002)	0.651 (0.056)	0.038 (0.002)	0.578 (0.051)
	Jitter	0.040 (0.001)	0.614 (0.053)	0.038 (0.002)	0.552 (0.054)
	JDE	0.042 (0.001)	0.629 (0.049)	0.041 (0.001)	0.628 (0.055)
Thyroid	Basic	0.045 (0.001)	0.074 (2e-4)	0.042 (0.001)	0.073 (0.002)
	Dither	0.045 (0.001)	0.074 (2e-4)	0.042 (0.001)	0.073 (0.002)
	Jitter	0.045 (0.001)	0.074 (1e-4)	0.042 (0.002)	0.073 (0.002)
	JDE	0.045 (0.001)	0.074 (2e-17)	0.043 (0.001)	0.074 (0.001)

Table 4: Results of ANN weight optimization using eight DE variants with tuned parameters. Table shows the average MSE and CLS obtained over 50 runs. Numbers in parentheses are the standard deviations. For each ANN dataset the best results are printed in bold-face.

Problem	MSE	CLS
Cancer	0.034 (0.003)	0.035 (0.002)
Card	0.107 (0.002)	0.143 (0.002)
Gene	0.072 (0.004)	0.133 (0.010)
Soybean	0.030 (0.001)	0.488 (0.030)
Thyroid	0.047 (3e-5)	0.074 (3e-17)

Table 5: Optimization of ANN weights using BP. Table shows the average MSE and corresponding CLS obtained over 50 runs. Numbers in parentheses are the standard deviations.

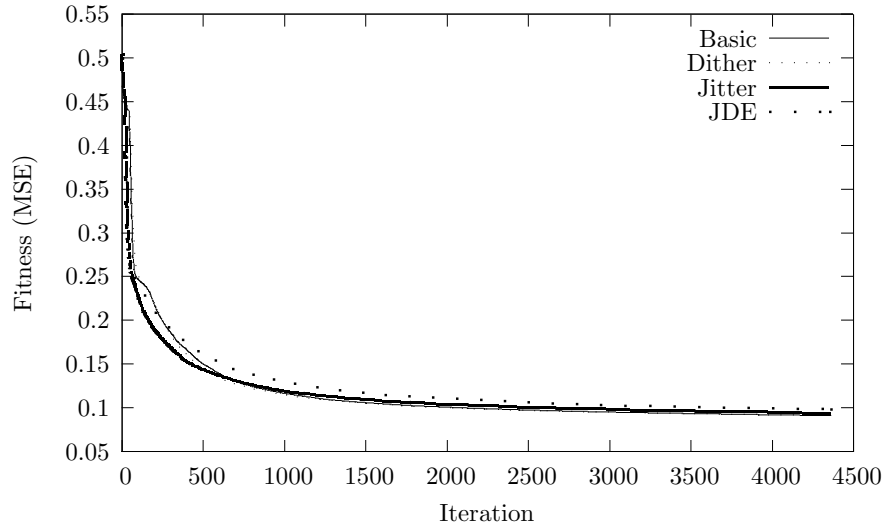


Figure 4: Fitness traces from using DE/best/1/bin/simple variants to optimize the ANN **Card** problem. Plot shows the average fitness obtained at each iteration of 50 optimization runs.

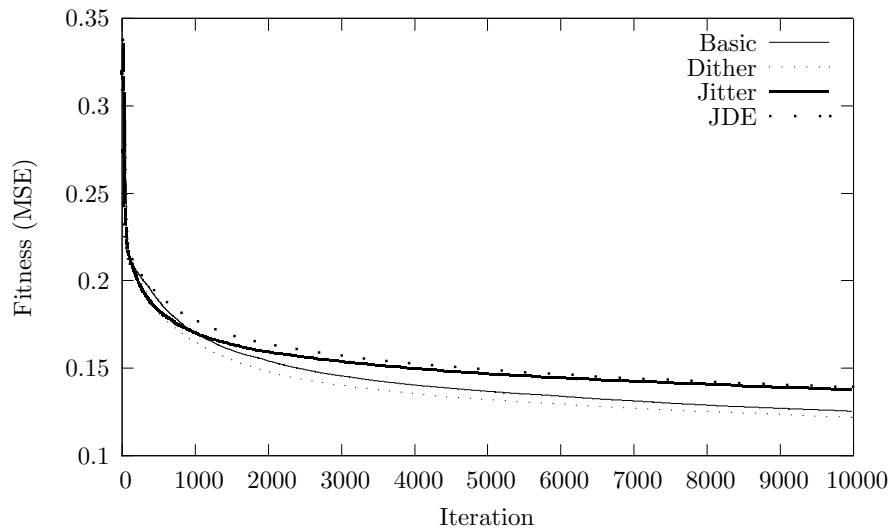


Figure 5: Fitness traces from using DE/best/1/bin/simple variants to optimize the ANN **Gene** problem. Plot shows the average fitness obtained at each iteration of 50 optimization runs.

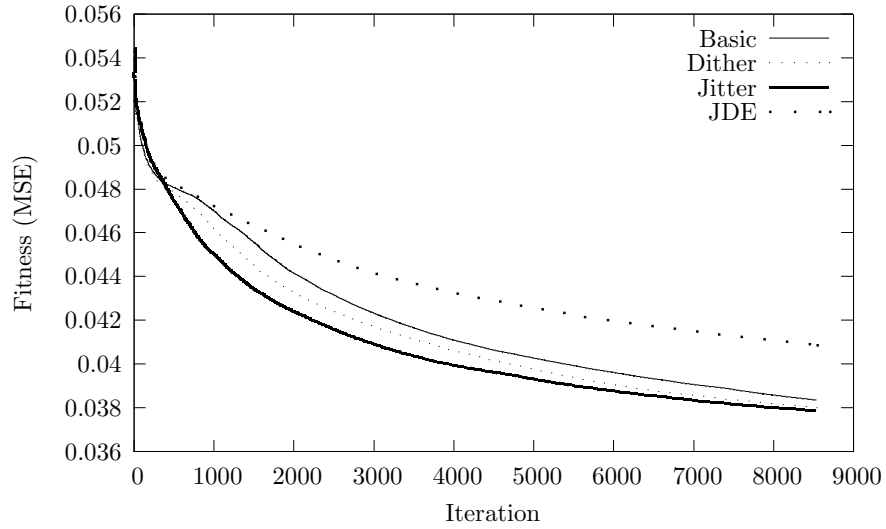


Figure 6: Fitness traces from using DE/best/1/bin/simple variants to optimize the ANN **Soybean** problem. Plot shows the average fitness obtained at each iteration of 50 optimization runs.

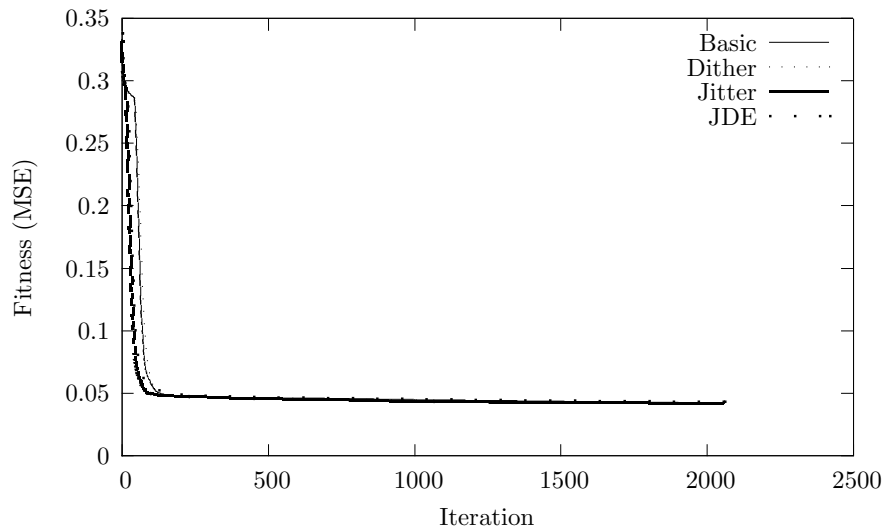


Figure 7: Fitness traces from using DE/best/1/bin/simple variants to optimize the ANN **Thyroid** problem. Plot shows the average fitness obtained at each iteration of 50 optimization runs.