Neural Networks 23 (2010) 568–582

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

2010 Special Issue

Meta-learning approach to neural network optimization

Pavel Kordík*, Jan Koutník[1], Jan Drchal, Oleg Kovářík, Miroslav Čepek, Miroslav Šnorek

Department of Computer Science and Engineering, FEE, Czech Technical University, Prague, Czech Republic

ARTICLE INFO

Article history:
Received 7 September 2009
Revised and accepted 3 February 2010

Keywords:
Neural network optimization
Ant colony
Evolutionary algorithms
Meta-learning
Meta-optimization
Optimization algorithm benchmark
Diversity promotion
Inductive modeling
Self-organization

ABSTRACT

Optimization of neural network topology, weights and neuron transfer functions for given data set and problem is not an easy task. In this article, we focus primarily on building optimal feed-forward neural network classifier for i.i.d. data sets. We apply meta-learning principles to the neural network structure and function optimization. We show that *diversity promotion*, *ensembling*, *self-organization* and *induction* are beneficial for the problem. We combine several different neuron types trained by various optimization algorithms to build a supervised feed-forward neural network called Group of Adaptive Models Evolution (GAME). The approach was tested on a large number of benchmark data sets. The experiments show that the combination of different optimization algorithms in the network is the best choice when the performance is averaged over several real-world problems.

1. Introduction

The search for optimal neural network for a given problem and data set is an increasingly complex task. Initially, optimization of connection weights only in Perceptron type networks was taken into account (Rosenblatt, 1957). Structural limitations of the topology impose limits on the number of parameters and the search space size. Additionally, considering different types of transfer functions may help the network to better fit (model) the input data (Murre, 1992). Finally, making the network topology non-regular may lead to simple and sparse resulting network but the complexity of search for the network is increased again.

In datamining, tasks and corresponding data sets are usually either of i.i.d. or sequential manner. Let us leave the sequential data (for whose processing recurrent neural networks are proper networks to choose) aside and concentrate to modeling of static i.i.d. data. In the field of datamining with the use of neural network, the feed-forward neural networks are usually considered.

In this paper, we propose a feed-forward neural network model called *Group of Adaptive Models Evolution (GAME)* (Kordík, 2006a, 2006b, 2009; Kordík, Kovářík, & Šnorek, 2007). The main goal is to show that inductively trained neural network model can benefit from diversity promotion, ensembling and self-organization.

The GAME network utilizes combination of several optimization methods to train the neurons within the network. It remains to be shown that such a combination gives consistently good results on a wide range of data sets.

The paper is organized as follows. Section 2 gives an overview of related work such as gradient approaches to neural network training, direct search through neuro-evolution and meta-learning. Section 3 introduces the GAME framework. Section 4 benchmarks underlying optimization methods. Section 5 evaluates the combination of optimization methods experimentally. Section 6 evaluates GAME performance on combination of benchmarks. Finally, Section 7 concludes the paper.

2. Related work

2.1. Gradient-based approach to neural network training

Training of neural network with layered regular topology having unified sigmoidal units is not straightforward. Gradient updates of the weights using back-propagation (Tsoukalas & Uhrig, 1996) of the error from layer to layer tend to stuck in the local optima as well as many iteration is needed until convergence is reached.

To overcome these problems, more efficient optimization techniques have been employed. Momentum (Tsoukalas & Uhrig, 1996) helps to overcome stuck in local optima, Quick propagation (Fahlman, 1988) and Resilient propagation (Riedmiller & Braun, 1992) algorithms go even further reducing the number of iterations needed.

The Cascade Correlation Algorithm (CCA) (Fahlman & Lebiere, 1991) employed induction to construct the network layer by layer,

* Corresponding author.
E-mail address: kordikp@fel.cvut.cz (P. Kordík).
[1] Jan Koutník is currently with IDSIA, SUPSI/USI, Manno-Lugano, Switzerland.

neuron by neuron. Instead of a predefined fixed topology, this network adapts to a particular problem. This freedom of choice in a network topology further increases complexity of the optimization task. The topology of CCA is still restricted to fully connected neurons.

One of the GAME algorithm challenges is to evolve feed-forward networks layer by layer using heterogeneous neurons. The topology of the network is not restricted, transfer functions of neurons are diverse, as well.

2.2. Neuro-evolution

Leaving the gradient weight optimization aside, direct search for the neural network parameters allows to search optimal weights, topology and possibly transfer functions setup. Evolutionary techniques are commonly used and the approach is called *neuro-evolution*. In Drchal, Kučerová, and Němeček (2002) a fixed topology feed-forward neural network weights are evolved with real-value encoded genomes. The methods which optimize not only parameters (weights), but also the topology, are called Topology and Weight Evolving Artificial Neural Networks (TWEANN). On top of traditional TWEANN approach, the GAME uses neurons with different transfer functions. Therefore, we extend the topology optimization term with appropriate transfer function selection.

There are basically two approaches to encode neural network weights and/or topology. *Direct* encoding evolves a set of weights directly expressed by real valued numbers. *Indirect* (or *developmental*, *generative*) encoding reduces the effort to enumerate the weights with their description by, generally speaking, a program, that computes the weights (topology and transfer function) (Schmidhuber, 1997). The program is the subject for the evolution.

In the field of the direct encoding, Angeline, Saunders, and Pollack (1994) proposes an algorithm called GeNeralized Acquisition of Recurrent Links (GNARL). In this approach Genetic Programming (GP) (Cramer, 1985; Koza, 1992; Schmidhuber, 1987) is used to add and remove links within possibly recurrent neural networks with a single hidden layer and a fixed number of neurons. Yao and Liu (1997) evolves directly encoded feed-forward neural networks featuring learning at the level of individuals. Particularly generated networks are adapted to the task using standard learning algorithms. Moriarty and Miikkulainen (1997) introduce Symbiotic, Adaptive Neuro-Evolution (SANE), in which population of neurons and population of networks randomly constructed from these neurons are coevolved simultaneously. The main drawback — the neurons can change their function by being placed to random positions in the network is fixed in Enforced SubPopulations (ESP) (Gomez & Miikkulainen, 1996), where each neuron for a particular position in a network is chosen from a different sub-population. A hierarchical variant of ESP (H-ESP) (Gomez & Schmidhuber, 2005) keeps a cache of best combinations of neurons, thus the coevolution runs at two different levels — neurons and networks. The state-of-the-art direct encoding algorithm called Cooperative Synapse Neuroevolution (CoSyNE) (Gomez, Schmidhuber, & Miikkulainen, 2008) performs probabilistic permutations of weights of coevolved synapses. CoSyNE has been shown to outperform all previously described neuroevolution techniques and is comparable to general parameter search methods like CMA-ES (Hansen & Ostermeier, 2001) applied for network weight learning on standard non-linear benchmarks such as a double pole balancing task.

NEAT (NeuroEvolution of Augmenting Topologies) (Stanley & Miikkulainen, 2002) is another TWEANN example. NEAT introduced a concept of historical markings, which are gene labels allowing effective genome alignment in order to facilitate crossover-like operations. Moreover, historical markings are used for computation of a genotypical distance between two individuals effectively organizing the population into niches. NEAT features so called *complexification*. It starts with the simplest network and gradually adds nodes and optimizes the connection weights.

In the field of indirect encoding Gruau's Cellular Encoding (CE) (Gruau, 1994) inspired by embryogenesis utilizes Genetic Programming to evolve programs that create neural network structure and weights. Stanley (2007) employed NEAT generated network as a function that computes weights of connections between neurons placed in a cartesian grid. The network called the Compositional Pattern-Production Network (CPPN) takes usually four dimensional vector of neuron coordinates as its input and computes the corresponding weight. The four-dimensional hypercube of neurons called *substrate* gave the approach the name HyperNEAT (D'Ambrosio & Stanley, 2007). The NEAT first generates flat weighted networks gradually increasing a variety of the weight matrices producing more complex network behaviors. Buk, Koutník, and Šnorek (2009) substitutes the NEAT in HyperNEAT with Genetic Programming reaching faster convergence by better GP exploration abilities. Hypercube encoding of neural network can utilize geometrical properties of the given task and is being applied to various number of tasks (D'Ambrosio & Stanley, 2008; Gauci & Stanley, 2007) featuring a reasonable scalability of the network without increasing the search space (size of the CPPN). A limitation of this state-of-the-art indirect encoding algorithm is that even for small network the CPPN or GP expression can become quite complex.

2.3. Optimization and meta-optimization

In neural networks we can optimize both continuous (weights) and discrete (topology) parameters. *A Continuous optimization* can be viewed as looking for extremes of N dimensional continuous function (parameters space). For such functions, an analytic gradient and a Hessian can often be computed. This information is used by many optimization algorithms to reach sufficient optima in a reasonable time (number of function evaluations). The analytic gradient or Hessian, for whatever reason, is not available, the search is performed numerically. For neural networks with standard sigmoidal neurons, an analytic gradient of the training error can be computed.

A Discrete optimization (and an associated *combinatorial optimization*) solves problems in which parameters are restricted to assume discrete values only. In a such search space, it is much more difficult to locate optimal solutions. These problems can be efficiently solved by algorithms tailored to a particular problem (or reduced to a problem within the same domain). There is no straightforward answer to the question which algorithm should be used for given problem and how to configure it.

Meta-heuristics (Engelbrecht, 2002; Glover & Kochenberger, 2003) combine several optimization algorithms to solve a single optimization problem more efficiently or to get a more robust algorithm for a wider range of problems. Meta-heuristics such as Simulated Annealing (SA), Evolutionary Algorithms (EA), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO) and Tabu Search have been successfully applied to many difficult optimization problems for which no satisfactory problem specific solution exists. However, expertise is required to adopt a meta-heuristic for solving a problem in certain domain.

Meta-optimization introduce a novel approach for search and optimization. A meta-optimization method operates on a top of a set of (meta-)heuristics. The most appropriate heuristic is determined and applied automatically by the technique prior or at each step of an optimization to solve a given problem. Meta-optimizations are therefore assumed to be problem independent and can be easily utilized by non-experts as well. Example strategies are pure random or random initial low-level heuristic with switching to other one after a descent into a local optima.

2.3.1. Optimization of optimization algorithm

Probably the simplest approach to meta-optimization is based on a two-level scheme. The first level is the *base-level algorithm*. The base-level algorithm works in search space of optimized problem and is looking for optimal solution of the given problem. The second level is *meta-level algorithm*, which optimizes parameters of base-level algorithm.

The meta-optimization approach is often used in a conjunction with Evolutionary Algorithms (EAs). If both base-level and meta-level algorithms are EAs, a term *meta-evolution* is often used. Optimized parameters can be: mutation rate, crossover rate, population size, etc. Typical representatives can be found in Diosan and Oltean (2007), Fogel, Fogel, and Atmar (1991) and Grefenstette (1986). The base-level EA parameters can be either *tuned* or *controlled* (Eiben, Michalewicz, Schoenauer, & Smith, 2007). The parameter tuning fixes the parameters for the whole base-level EA run. On the other hand, the parameter control continually adjusts parameters throughout the evolution. The controlled parameters can be either *adapted* using a predetermined rule or a *self-adaptation* can be employed (Deugo & Ferguson, 2004). Here, the parameters to be adapted are encoded into individual chromosomes and undergo mutations and recombinations. Other good source of information concerning adaptation can be found in Smith and Fogarty (1997).

Memetic algorithms (MA) are evolutionary based algorithms that apply a local search process to refine solutions of hard problems (Hansen, 2004) — MAs are a particular class of global-local search hybrids. Traditionally, MAs with Lamarckian learning procedures are based on the use of a evolutionary algorithm and a single local search method for local improvements (Ong & Keane, 2004).

Meta-Lamarckian learning in MAs uses a set of several local search methods, choosing the proper method for given problem to achieve an improved search performance. A very similar approach to the meta-evolution based on Particle Swarm Optimization can be found in Meissner, Schmuker, and Schneider (2006). Another method in Yuan and Gallagher (2007) is suitable for an optimization of both numeric and symbolic parameters.

2.3.2. Recommendation or combination of base-level algorithms

All above mentioned meta-optimization algorithms were limited to find optimal parameter settings for a base-level algorithm only.

Even more sophisticated strategy is to *combine* multiple methods together. *Switching* optimization algorithms in periods of time is the obvious possibility.

In Satoru Hiwa and Miki (2007), authors switch three optimization algorithms: DIRECT (DIviding RECTangles), GA (Genetic Algorithm) and SQP (Sequential Quadratic Programming). The algorithms are run sequentially starting with the DIRECT global search method, then improving accuracy and narrowing down the search area by GA and finally finishing with fine-tuning SQP. The switch to a more local-oriented search is realized when a predefined termination criterion is met. These criteria involve: limits on a number of iterations, diversity measure or getting close enough to the known value of the optimum.

MAGMA (Roli & Milano, 2004) is a MultiAGent Metaheuristics Architecture conceived as a conceptual and practical framework for meta-heuristic algorithms. Authors revisit meta-heuristics in a multiagent perspective and define agents of a different level of abstraction. Level-0 agents provide feasible solutions for upper levels; Level-1 agents deal with solution improvements and provide local search; Level-2 agents have global view of the search space, or, at least, their task is to guide the search towards promising regions; Level-3 is introduced to describe higher level strategies like cooperative search and any other combination of meta-heuristics.

2.3.3. Methods extracting a meta-level information

The important property of the above mentioned algorithms is, that they do not extract and exploit any information concerning the base-level optimization runs or information describing the problem being solved. Further, we will use the term *meta-data* for this kind of information. Examples of meta-data are statistical properties of attributes, mutual information, information entropy, etc. Meta-data may even contain data about progress of base-level optimization, for example best-so-far solution, number of steps since last best-so-far solution update, etc.

Algorithm recommendation (Brazdil, Giraud-Carrier, Soares, & Vilalta, 2009) methods, on the other hand, are given a set of optimization algorithms out of which they try to select (using meta-data) a single best optimization algorithm to solve the problem.

In Merz and Freisleben (2000), such meta-data extraction is proposed. Authors perform an analysis of a fitness landscape based on an occurrence of local extremes. They employ an auto-correlation and a fitness distance correlation analysis.

STAGE algorithm (Boyan, 1998) uses the original objective function value and a local search algorithm. During the first stage, the algorithm follows a local search in the original fitness landscape. When a local optima is detected, STAGE uses a machine learning method (Artificial Neural Networks, Linear Regression, Temporal Difference) to learn a value function mapping of a local search start state to the fitness value of the expected local optima encountered by a search beginning in that state. In the second stage it applies the local search to the new fitness landscape given by the learned model. Both stages are then run repeatedly. Although STAGE can be used for both discrete and continuous optimization problems, the experiments were focused on discrete tasks: bin-packing, VLSI routing or Boolean Satisfiability.

In Cicirello (2003), AQDF tool, which can be used to model a distribution of solutions given by a single iteration run of stochastic search algorithms, is introduced. AQDF leads to a search control framework, called QD-BEACON, which employs online-generated statistical models of a search performance to effectively combine search heuristics. Again, the approach focuses on discrete problems.

In Aine, Chakrabarti, and Kumar (1992), a meta-level controller was introduced. It takes decisions for both optimization algorithm parameter settings and the time allocated for the optimization. The controller involves profiling of Simulated Annealing and Genetic Algorithm.

In Burke, Petrovic, and Qu (2006), a novel approach of meta-optimization is presented. The algorithm extracts an optimization meta-data while solving different problems. The meta-data are used to create a *knowledge-base* (implemented by Case Based Reasoning). The knowledge-base is later used to select a proper heuristic in order to solve a new previously unseen problem. A drawback of the approach lies in a limitation to solve two preselected discrete optimization tasks only: a course timetabling and an exam timetabling — the optimization meta-data are problem dependent (uses features like number of courses, number of time periods, number of rooms, etc.).

2.3.4. Combination of models in meta-learning

In the meta-learning, models are usually combined together to obtain better or more reliable results. This technique is often referred as an *ensembling*. Some of these approaches are adopted for meta-optimization.

In *bagging* algorithm several training subsets are created from the original training set. A classifier is created for each subset. The result for an unknown instance presented to classifiers is calculated as an average or a majority of their responses (Brazdil et al., 2009; Breiman, 1996; Freund & Schapire, 1995).

In *boosting* (Brazdil et al., 2009; Schapire, 1990) algorithm there is a single classifier in the beginning. In regions where it fails to respond correctly, the instances are boosted — their weight is increased and a second classifier is created. The second classifier is expected to overlook well-classified instances and to focus on misclassified instances of the first classifier. In the regions where the second classifier failed, weights of instances are increased again and the third classifier is introduced and so on.

A *stacking* (Brazdil et al., 2009; Wolpert, 1992) uses several different models trained on a single training set. Responses of these models represent meta-data which are used as inputs of a single final model which finally decides on the correct class.

Cascade generalization (Brazdil et al., 2009; Gama & Brazdil, 2000) presents a sequential approach to combination of models. In Cascade Generalization a sequence of learners is created. Inputs of learner A_i consist of input base-data and also outputs (class probabilities) of previous classifier A_{i-1}. Thus each classifier in the sequence is partially base-learner and partially meta-learner. Newly classified instances pass through the sequence and output of the last classifier becomes the output of the whole ensemble.

Cascading (Alpaydin & Kaynak, 1998; Brazdil et al., 2009; Kaynak & Alpaydin, 2000) is related to boosting. As boosting, it changes distribution over the training instances. In the beginning all instances have the same weight. Then the first model is introduced with the first-level base-learner. After that, all instances are reweighted according to confidence of the first-level learner. Then second learner is created and instances are reweighted again and so on until all instances are correctly classified with desired confidence or defined number of classifiers is reached. When classifying the new instance, the instance is passed through the sequence of classifiers. The first classifier whose response reaches a specified threshold is the output of the whole system.

Delegating (Brazdil et al., 2009; Ferri, Flach, & Hernández-Orallo, 2004) presents a cautious approach to the classification. There is a sequence of classifiers and if output of a classifier A_i does not meet a predefined threshold the instance is passed to a next classifier and so the final decision is delegated to other classifier. It is alike cascading but in cascading all instances are processed by all classifiers. In delegating, instances are processed by a classifier only if previous classifiers lack the confidence in their results.

Arbitrating (Brazdil et al., 2009; Ortega, 1996; Ortega, Koppel, & Argamon, 2001) is similar to the delegating. The basic idea is the same as for delegating — different classifiers have different areas of expertise. But unlike delegating the areas of expertise are determined in run-time. All classifiers in ensemble are trained on full, unmodified training set. When unknown instance is presented to the system, all classifiers present their output and the confidence. The classifier with the highest confidence is selected to produce the final result.

The area of confidence of classifiers is determined by a referee. The referee is typically a decision tree that predicts which of the models responds correctly for a given input data.

Meta-learning models or classifiers have to be diverse, otherwise their combination would not improve the quality of the prediction. Next section maps the diversity in the neural networks domain.

2.4. Diversity and neural networks

Diversity of individual neurons plays critical role in neural networks domain. Each training algorithm has some strategy to enforce the diversity of neurons. For most of the algorithms, the primary goal is to reduce the generalization error of network and diversity of neurons is a side product.

The diversity in a neural network can be promoted by

- data set manipulation:
  - employed input features,
  - set of instances used for training;
- neuron manipulation:
  - distance of neuron weights (or coefficients),
  - type and complexity of transfer function.

The goal is that neurons exhibit diverse errors.

In RBF networks, individual neurons are specialized on predicting part of the space. In boosting, something similar happens, i.e. the individual models in the cascade try to correct the errors made by its predecessors.

In Ritchie, White, Parker, Hahn, and Moore (2003), genetic programming is used to design multilayered perceptron network. The article also suggests to promote diversity among individual solutions by means of the island strategy. This is consistent with Stanley (2004), where fitness sharing is used to protect newly emerged diverse networks while they evolve their weights.

Very efficient and straightforward solution, presented in this article, is to train individual neurons using varied architectures, instances and features in order to obtain implicit diversity. We also experiment with negative correlation (Brown, 2004), where the error function of neurons is manipulated in order to train non-correlated neurons within the network.

2.5. Self-organization and induction

The self-organization (Muller & Lemke, 2000) is another important principle that can help to build unbiased models from diverse data sets. Self-organized neural network,[2] in our sense, is able to adapt itself to a data set complexity without any external information. Self-organizing networks are often constructed inductively from data. The induction means gathering small pieces of information, combining it, using already collected information in the higher abstraction level to get complex overview of the studied object or process.

Inductive modeling methods utilize the process of induction to construct models of studied systems. The construction process starts from a minimal form and the model grows until a system complexity is matched.

A problem is decomposed into small subproblems, by combining subsolutions we get higher-level solution and so on. A modeling problem can be decomposed into subproblems by scaling:

- *Input features* — number of input features used by neuron can be limited.
- *Instances* — subset of instances used by neuron can be limited.
- *Transfer function complexity* — complexity of transfer function can be limited.
- *Parameters maximum* — maximal values of parameters can be limited (weight decay).
- *Acceptable error* — parameter optimization can be stopped prematurely.

Our approach decomposes a problem by scaling number of input features (maximum 1 input for neurons in the first layer of network). Also complexity of the transfer function increases with increasing number of neurons layer. The subset of instances used for training can be increased with network layer. Acceptable accuracy can be also increased with network layers. There are many more ways how to decompose the problem and build an inductive self-organized network. The question is whether it is more efficient to scale all items (from the list above) at once and

[2] Do not confuse Self-organized neural network with Self-Organizing Map.

finish with a single network, or to scale them one by one finishing e.g. by ensemble of ensembles of networks of neurons. This has to be investigated in future.

The optimization of a hybrid neural network for high dimensional data set is extremely difficult task. Self-organization and induction helps us to explore huge space of all possible neural network topologies with higher efficiency.

2.6. Optimization of neural network for arbitrary problem

Our motivation is to develop an optimization strategy capable of discovering suitable neural network topology and weights for arbitrary data set. By "suitable" neural network, we mean a network with reasonable good generalization error.

The problem is that for arbitrary data set, any assumption on data distribution and properties could not be done. An universal strategy is needed for neural network optimization.

In connection with the "universal strategy" we have to mention the well-known No Free Lunch Theorem (Burke & Kendall, 2005; Wolpert & Macready, 1997). It states, that a general-purpose universal optimization strategy is theoretically impossible, and the only way one strategy can outperform another is in a specialization to a specific problem under consideration (Ho & Pepyne, 2002). However, there are reasons why we should not take this interpretation literally. At first, humans often have a little prior knowledge to work with. The second, incorporating a prior knowledge does not give much of a performance gain on a considerable range of problems. The third, a human time is very expensive relatively to a computer time. There are many cases in which a company would choose to solve a problem slowly with an unmodified computer program rather than rapidly with a human-modified program. And finally, we have to point out, that although we want to be able to solve largest possible set of problems, all these problems will likely have a real-world origin and possibly common properties. This set forms just a tiny subset of all possible problems, out of which the majority is represented by a randomized fitness landscape (Burke & Kendall, 2005).

Poli, Graff, and McPhee (2009) prove that NFL does not hold for a common task of symbolic regression by GP (Genetic Programming). Similar results hold also for neural networks (Poli & Graff, 2009).

Optimization algorithms come in many flavors — numerical, nature inspired, deterministic, stochastic, etc. In this article we combine strengths of many different algorithms into a single meta-optimization strategy. This strategy is a significant step towards *data driven optimization of neural network.*

3. Optimization strategy in the GAME neural network

Group of Adaptive Models Evolution (GAME) (Kordík, 2009) is a method loosely based on the Group Model Data Handling (GMDH) theory (Ivakhnenko, 1971). GMDH represents the inductive approach in modeling. It was designed to automatically generate a model of a system in a form of polynomial equations. The most popular GMDH algorithm is the Multilayered Iterative Algorithm (MIA GMDH).

To simplify the optimization procedure, GMDH neurons possess an unified polynomial transfer function (e.g. $y = a_1x_1^2 + a_2x_2^2 + a_3x_1x_2 + a_4x_1 + a_5x_2 + a_6$). A number of inputs is restricted to two (x_1, x_2) and the parameters of neurons are optimized by a single step Least Mean Squares (LMS) method (Ivakhnenko & MMüller, 1995). The advantage of the algorithm is that the structure of MIA GMDH networks is not fixed (pre-defined) as in case of traditional MLP networks. The network grows (adds layers of neurons) until an optimal complexity is reached on a particular data set. In each layer all possible pairwise combinations of inputs are examined.

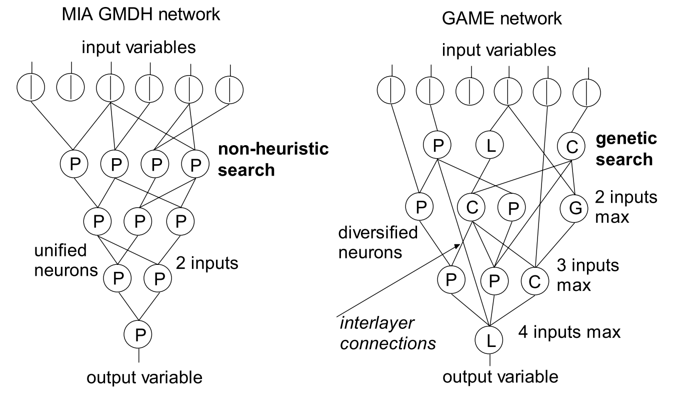


Fig. 1. Examples of typical neural network topologies produced by the MIA GMDH algorithm (left) and the GAME algorithm (right). Unlike MIA GMDH, GAME allows different types of neurons (units), non-constant number of neuron inputs and interlayer connections.

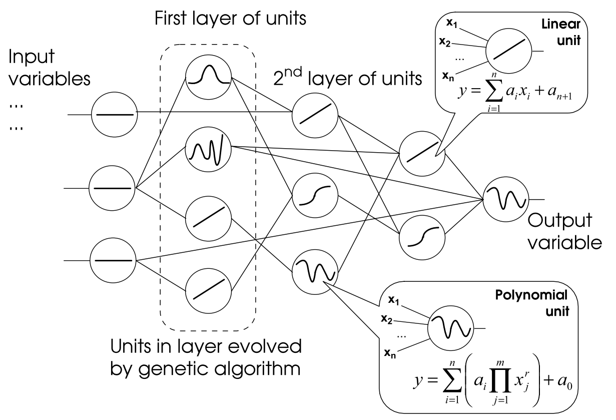

First layer of units
Input variables
...
...
2nd layer of units
Linear unit
$y=\sum_{i=1}^{n} a_i x_i + a_{n+1}$
Output variable
Polynomial unit
$y=\sum_{i=1}^{n}\left(a_i\prod_{j=1}^{m} x_j^r\right)+a_0$
Units in layer evolved by genetic algorithm

Fig. 2. The structure of the GAME model is evolved layer by layer using a Niching Genetic Algorithm. During the training of a neuron, a selected optimization method adjusts coefficients $(a_1, \ldots, a_n)$ in the transfer function of the neuron.

Most of them are suppressed, just a few candidates are selected according to an external criteria.

A disadvantage of the MIA GMDH is that there are still some restrictions of the topology (neurons with two inputs only and predefined polynomial form can have connections to previous layer only). The most problematic is inefficient full state space search (all possible neuron candidates are examined).

The GAME network (see Fig. 1) has more degrees of freedom (neurons with more inputs, interlayer connections, different transfer functions etc.) than the MIA GMDH network.

An example of an inductive model created by the GAME algorithm is depicted in Fig. 2. Similarly to the GMDH MIA and also to a Multi-Layered Perceptron (MLP) neural networks, GAME units (neurons) are connected in a feed-forward network (model). A hybrid model can be composed of neurons with different transfer function types (e.g. sigmoid, polynomial, sine, linear, exponential, rational, etc.).

Thanks to sigmoidal units, GAME network can be used as classifiers (polynomial transfer function neurons are bad in expressing a decision boundary). For multi class problems, we build one GAME model for each class and the predicted class is given by model with highest response.

As you can see, the GAME algorithm lifts many restrictions of topology, such as unified polynomial neurons, two connections to preceding layer, etc. It is not feasible any more to search a full state space of all possible topologies. Therefore we have introduced a heuristics to optimize networks in a more efficient way. The heuristics has to navigate in a space of possible network topologies

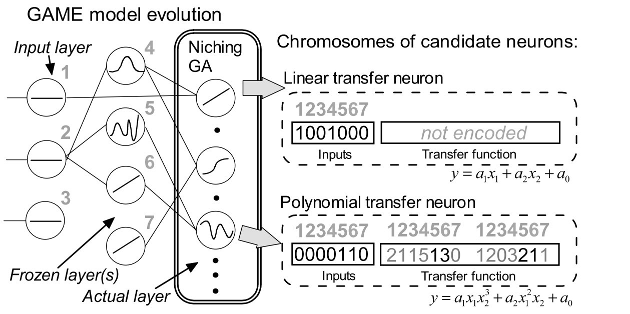


Fig. 3. Neurons are evolved by the DC algorithm. Their chromosomes contain inputs, for some types we encode also structure of the transfer function or optimization method used to estimate the coefficients.

(connections) and parameters (coefficients). The optimization problem is therefore discrete and continuous at the same time. The GAME method can be classified as a TWEANN method (see Section 2.2).

A model is built layer by layer, similarly to the MIA GMDH. Neurons in previous layers are "frozen" (Fig. 3), they are not modified any more. All neurons are evaluated using RMS error, i.e. using the differences of the target signal and the actual neuron output. This means, that all the neurons are trained to produce the same output and it is not in accordance with our previous statements on role of diversity in neural networks. The diversity is hidden in a way, such that neurons reduce a residual error. Neurons from second and deeper layers have normally one or more inputs connected to neurons from previous layers and their aim is to reduce the residual error between target output and output preprocessed by its input neurons. Furthermore, diversity is promoted by different training data of neurons (subsets of features and instances) and also by means they are evolved.

Neurons in each layer are optimized by a Niching Genetic Algorithm. Niching methods (Mahfoud, 1995) extend GAs to domains that require location of multiple solutions. They promote the formation and maintenance of stable subpopulations in GAs which supports diversity in solutions. The GAME algorithm uses the *Deterministic Crowding* (DC) (Mahfoud, 1995) niching method to optimize inputs of neurons and the structure of their transfer functions. An example of encoding into chromosome is in Fig. 3. The transfer functions are crossed over only when genomes of neurons are compatible. The diversity (or distance) of neurons is computed as the average of their genotypic and phenotypic distances (difference in inputs, transfer function and inverted correlation of errors) (Kordík, 2006b). From the population of neurons, evolved by DC algorithm, we select one representant from each niche. These neurons remains fixed and act as preprocessing units for neurons in subsequent layers choosing them as their input connections.

Parameters of the model (coefficients of neurons transfer functions) are optimized independently (Kordík et al., 2007). Another diversity promotion in GAME is that different types of neurons are assigned different continuous optimization methods to find their coefficients.

The primary topic of this article is the optimization of coefficients (for example $a_1, \ldots, a_n$ in Fig. 2).

3.1. Optimization of coefficients (weights)

The goal is to find optimal values of coefficients $a_1, a_2, \ldots, a_n$ in transfer functions of GAME neurons (see Fig. 4).

The optimal values of parameters are values minimizing the difference between a behavior of a real system and its model. This difference is typically measured by the root mean squared error (RMS).

The error of a neuron on training data set is the sum of output differences from target variable for all training vectors:

$$E = \sum_{j=0}^{m} \left(y_j - d_j\right)^2, \tag{1}$$

where y_j is the output of the neuron for the jth training vector and d_j is the corresponding target output value.

Only for neurons with linear or polynomial transfer function a single step LMS method can be used. For more complex transfer functions, the optimization of coefficients is an iterative process. In each iteration, the optimization method proposes values of coefficients and the GAME neuron returns its error on training data with these proposed coefficients (see Fig. 4(a)). If the analytical gradient of the error can be computed, the number of iterations would be significantly reduced, because the optimization method is informed in which direction coefficients should be adjusted (see Fig. 4(b)). The gradient of the error $\nabla\vec{E}$ in the error surface of neuron can be written as:

$$\nabla\vec{E} = \left(\frac{\partial\vec{E}}{\partial a_1}, \frac{\partial\vec{E}}{\partial a_2}, \ldots, \frac{\partial\vec{E}}{\partial a_n}\right), \tag{2}$$

where $\frac{\partial\vec{E}}{\partial a_i}$ is a partial derivative of the error in the direction of the coefficient a_i. It tell us how to adjust the coefficient to get smaller error E on the training data. This partial derivative can be computed as:

$$\frac{\partial\vec{E}}{\partial a_i} = \sum_{j=0}^{m} \frac{\partial\vec{E}}{\partial y_j} * \frac{\partial y_j}{\partial a_i}, \tag{3}$$

where m is the number of training vectors. The first part of the summand can be is easily derived from the Eq. (1) as:

$$\frac{\partial\vec{E}}{\partial y_j} = 2\sum_{j=0}^{m} \left(y_j - d_j\right). \tag{4}$$

The second part of the summand from the Eq. (3) is unique for each neuron, because it depends on its transfer function. We demonstrate the computation of the analytic gradient for the Gaussian neuron. For the other neurons the gradient is computed in a similar way.

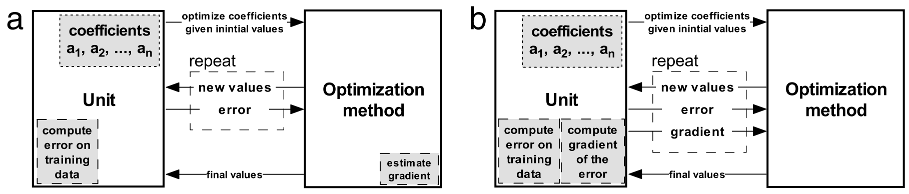


Fig. 4. Optimization of the coefficients can be performed without an analytic gradient (a) or with the gradient supplied (b). The optimization method is initialized by a starting point (vector of coefficients). Then, in each iteration, new coefficients are presented to the neuron and error on validation data is provided back to the optimization method.

Table 1
Number of evaluations saved by supplying gradient depending on the complexity of the error function.

Complexity energy fnc.	Avg. evaluations without grad.	Avg. evals. with grad.	Avg. gradient calls	Evaluations saved (%)	Computation time saved (%)
1	45.825	20.075	13.15	56.19	13.15
2	92.4	29.55	21.5	68.02	33.12
3	155.225	44.85	34.875	71.11	37.41
4	273.225	62.75	51.525	77.03	48.75
5	493.15	79.775	68.9	83.82	62.87

3.1.1. *The analytic gradient of the gaussian neuron*

The most common distribution in nature follows the gaussian probability density function $f(x) = \frac{1}{\sqrt{2\pi}\sigma} * \mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}}$. Neurons with gaussian transfer function are typically used in Radial Basis Function Networks. We have modified the function for our purposes. We added coefficients to be able to scale and shift the function, another coefficients are weighting input variables:

$$y_j = (1 + a_{2n+1}) * \mathrm{e}^{\underbrace{-\frac{\sum_{i=1}^{n}\left(a_i * x_{ij} - a_{n+i}\right)^2}{(1+a_{2n+2})^2}}_{\rho_j}} + a_0, \tag{5}$$

where n is number of inputs.

Note that we can also add coefficients to scale the hat in each dimension, etc.

Below, we derive the gradient of the error (see Eq. (2)) for the our version of the gaussian transfer function (Eq. (5)).

We need to derive partial derivatives of the error function according to Eq. (3). The easiest partial derivative to compute is the one in the direction of the a_0 coefficient. The second term $\frac{\partial y_j}{\partial \rho_j}$ is equal to 1. Therefore we can write $\frac{\partial \vec{E}}{\partial a_0} = 2\sum_{j=0}^{m}\left(y_j - d_j\right)$. In the case of the coefficient a_{2n+1}, the equation becomes more complicated

$$\frac{\partial \vec{E}}{\partial a_{2n+1}} = 2\sum_{j=0}^{m}\left[\left(y_j - d_j\right) * \mathrm{e}^{-\frac{\sum_{i=1}^{n}\left(a_i * x_{ij} - a_{n+i}\right)^2}{(1+a_{2n+2})^2}}\right]. \tag{6}$$

Remaining coefficients are in the exponential part of the transfer function. Therefore the second summand in the Eq. (3) cannot be formulated directly. We have to rewrite the Eq. (3) as

$$\frac{\partial \vec{E}}{\partial a_i} = \sum_{j=0}^{m}\left[\frac{\partial \vec{E}}{\partial y_j} * \frac{\partial y_j}{\partial \rho_j} * \frac{\partial \rho_j}{\partial a_i}\right], \tag{7}$$

where ρ_j is the exponent of the transfer function (5). Now we can formulate partial derivatives of remaining coefficients as

$$\begin{aligned}\frac{\partial \vec{E}}{\partial a_{2n+2}} &= 2\sum_{j=0}^{m}\left[\left(y_j - d_j\right) * (1 + a_{2n+1})\mathrm{e}^{\rho_j}\right. \\ &\quad \left. \times 2\frac{\sum_{i=1}^{n}\left(a_i * x_{ij} - a_{n+i}\right)^2}{(1+a_{2n+2})^3}\right]\end{aligned} \tag{8}$$

$$\begin{aligned}\frac{\partial \vec{E}}{\partial a_i} &= 2\sum_{j=0}^{m}\left[\left(y_j - d_j\right) * (1 + a_{2n+1})\mathrm{e}^{\rho_j}\right. \\ &\quad \left. \times -2\frac{a_i^2 * x_{ij}^2 - a_{n+i} * x_{ij}}{(1+a_{2n+2})^2}\right]\end{aligned} \tag{9}$$

$$\frac{\partial \vec{E}}{\partial a_{n+i}} = 2\sum_{j=0}^{m}\left[\left(y_j - d_j\right) * (1 + a_{2n+1})\mathrm{e}^{\rho_j} * -2\frac{a_{n+i} - a_i * x_{ij}}{(1+a_{2n+2})^2}\right]. \tag{10}$$

We derived the analytic gradient of error on training data for the Gaussian transfer function neuron. An optimization method often requests these partial derivatives every iteration to be able to adjust parameters in proper direction. This mechanism (as described on Fig. 4(b)) can significantly save the number of error evaluations needed.

3.1.2. *Analytic gradient benchmark*

We performed an experiment to evaluate the effect of analytic gradient computation. The Quasi-Newton optimization method (Dennis & Schnabel, 1996) was used to optimize the neuron with logistic transfer function. In the first run the analytic gradient was provided and in the second run, the gradient was not provided so the QN method had to estimate the gradient itself. We measured the number of function evaluation calls and for the first run, we recorded also the number of gradient computation requests.

The results are displayed in Table 1. In the second run, without the analytic gradient provided, the number of error function evaluation calls increased exponentially with rising complexity of the error function. For the first run, when the analytic gradient is provided, number of error function evaluation calls increases just linearly and the number of gradient computations grows also linearly. The computation of gradient is almost equally time-consuming as the error function evaluation. When we sum up these two numbers for the first run, we still get growth increasing linearly with the number of layer (increasing complexity of the error surface). This is superb result, because some models of complex problems can have 20 layers, the computational time saved by providing the analytic gradient is huge.

3.2. *Optimization methods to estimate parameters of neurons*

In the last experiment, the Quasi-Newton method was used to estimate parameters of neurons with the logistic transfer function. However many other gradient based optimization methods exist and there is no hint which method would be used. For some problems, the usage of analytic gradient can worsen a convergence characteristic of optimization methods (getting stuck in local minima). For such problems it is more appropriate to use other methods such as evolutionary algorithms, swarm or colony optimization methods.

The complexity of parameter optimization depends on many factors. First of all, the transfer function of actual neuron can be of different type and complexity. Then, all preceding neurons contribute to the complexity of optimization task. As demonstrated by previous experiment, neurons in the first layer require significantly less iterations until convergence compared to neurons in the next layer (complexity of error function increases layer by layer). The most significant factor is probably the complexity of the data set.

Table 2
Optimization methods implemented in GAME engine summary.

Abbrv.	Search	Optimization method
QN	Gradient	Quasi-Newton method
CG	Gradient	Conjugate gradient method
DE-pal	Evolutionary	Differential evolution ver. 1
DE	Evolutionary	Differential evolution ver. 2
SADE	Evolutionary	SADE genetic method
CMAES	Evolutionary	Covariance matrix adaptation ES
PSO	Swarm	Particle swarm optimization
PSOIW	Swarm	Particle swarm optimization IW
CACO	Colony	Cont. ant colony opt.
ACO*	Colony	Ext. Ant colony opt.
DACO	Colony	Direct ACO
AACA	Colony	Adaptive ant colony alg.
API	Colony	API ant alg.
HGAPSO	Hybrid	Hybrid of GA and PSO
SOS	Other	Stoch. orthogonal search
OS	Other	Orthogonal search ver. 1
OS-pal	Other	Orthogonal search ver. 2
RANDOM	Other	Random search

It is clear, that each neuron has a unique error surface and therefore it is not efficient to use one universal optimization algorithm.

There are many algorithms for continuous optimization we can choose from. In the GAME engine several different optimization algorithms are implemented. These methods play the role of base-learners and they are listed in the Table 2 and described in the next section.

The question "Which optimization method is the best for given neuron?" has not a simple answer. There is no method superior to others for all possible error surfaces. However, there are popular methods performing well on whole range of problems.

Among these popular methods, we can include so called gradient methods — the Quasi-Newton method, the Conjugate Gradient method and the Levenberg–Marquardt method. They use an analytical gradient or even a Hessian matrix (or their estimation) of a problem error surface. The derivatives brings them faster convergence, but in cases when the error surface is highly non-continuous, they are likely to get stuck in a local optima.

Other popular optimization methods are Evolutionary Algorithms. They search the error surface by sampling it with several individuals. Such search is usually slower, but more prone to get stuck in a local minima. The Differential Evolution (DE) perform evolutionary search with an improved crossover scheme.

The Particle Swarm Optimization (PSO) and the Ant Colony Optimization (ACO) are another nature inspired methods based on self-organization principles. While the PSO mimics the swarm behavior of bird flocks or fish schools, the ACO imitate real ants and their communication using pheromones.

Optimization methods with different behavior are often combined into a single algorithm such as the Hybrid of the Genetic Algorithm and the Particle Swarm Optimization (HGAPSO).

In the next section gives a short description of optimization methods used. Then we compare their performance and discuss the question how the best method for certain data set can be found. We also experiment with evolutionary choice of optimization methods (see Section 5). Finally, we summarize results over several data sets and recommend the best optimization strategy.

3.2.1. Gradient based methods

The *Quasi-Newton method* (QN) (Dennis & Schnabel, 1996) is a very popular method of non-linear continuous optimization. It computes search directions using first-order (gradient) and second-order derivatives (Hessian matrix). To reduce the computational complexity the Hessian matrix is not computed directly, but estimated iteratively using so called updates (Salane & Tewarson, 1980).

The *Conjugate Gradient* method (CG) (Wade, 2006), a non-linear iterative optimization algorithm, is based on an idea that the convergence can be improved by considering also all previous search directions, not only the actual one. Restarting (previous search direction are forgotten) often improves properties of CG method (Shewchuk, 1994). CG method uses only the first-order derivatives.

3.2.2. Evolutionary Algorithms

Evolutionary Algorithms (EAs) are inspired by Darwin's theory of evolution. EAs cover more different approaches, mainly the *Genetic Algorithms* (GAs) (Holland, 1975) and the *Evolution Strategies* (ES) (Rechenberg, 1973) Population of individuals are evolved according to simple rules of evolution. Each individual (phenotype) has a *fitness* assigned. Phenotype is constructed using a genetic information (genotype). Individuals are crossed and mutated by genetic operators while the most fit individuals are selected to survive. After several generations, the mean fitness of individuals stagnates.

The *Differential Evolution* (DE) (Storn & Price, 1997) is a Genetic Algorithm with a special crossover scheme. It adds a weighted difference between two individuals to a third individual. For each individual in the population, an offspring is created using the weighted difference of parent solutions. The offspring replaces the parent in a case it is fitter. Otherwise, the parent survives and is copied to the next generation. The pseudocode describing, how the offspring is created, can be found e.g. in Vesterstrom and Thomsen (2004). In our experiments we use two different implementations (DE, DE-pal (Drummond & Strimmer, 2001)) of the method. Comments on possible modification for combinatorial optimization can be found in Onwubolu and Davendra (2009).

The *Simplified Atavistic Differential Evolution* (SADE) algorithm (Hrstka & Kučerová, 2004) is an Evolutionary Algorithm using a real-valued encoding. It works with three genetic operators: the mutation (gaussian noise), the local mutation (final tuning) and the DE crossover scheme mentioned above.

The *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES) (Hansen & Ostermeier, 2001) is the state-of-the-art Evolution Strategy where new individuals are sampled according to a continually updated covariance matrix of a multivariate normal mutation distribution.

3.2.3. Swarm methods

The *Particle Swarm Optimization* method (PSO) uses a swarm of particles to locate an optimum. According to Juang and Liou (2004) particles "communicate" information they find about each other by updating their velocities in terms of local and global champions; when a new champion is found, the particles will change their positions accordingly so that the new information is "broadcasted" to the swarm. The particles are always drawn back both to their own personal best positions and also to the best position of the entire swarm. They also have stochastic exploration capability via the use of random constants. Here, we use to modifications of the algorithm: the canonical one (PSO) and the *Particle Swarm Optimization with Stochastic Inertia Weight* (PSO-IW) (de Oca, Stützle, Birattari, & Dorigo, 2006).

3.2.4. Colony optimization methods

The *Ant Colony Optimization* (ACO) algorithm is primarily used for discrete problems (e.g. Traveling Salesman Problem, packet routing). However many modifications of the original algorithm for continuous problems have been introduced recently (Tsutsui, Pelikan, & Ghosh, 2005). These algorithms mimic the behavior of real ants and their communication using pheromones. We have so far implemented the following ACO based algorithms:

The *Continuous Ant colony optimization* (CACO) was proposed in Bilchev and Parmee (1995) and it works as follows. There is an ant nest in a center of a search space with several search vectors leading from the nest. Ant chooses a direction using a roulette wheel selection applied to the pheromone amounts attached to each vector. From the position determined by the selected vector the ant performs a random walk inside a local search radius. This radius can shrink in time to do more detailed search around the point. The quantity of pheromone associated with explored vector is increased proportionally to the quality of the solution. If a better solution is found, the vector is changed to point to actual ant position.

The *Ant Colony Optimization for Continuous Spaces* (ACO*) (Blum & Socha, 2005) was designed for the training of feed-forward neural networks. However, it is able to solve mixed discrete-continuous optimization problems. In case of continuous variables the pheromone is represented by probability density function (PDF) — a mixture of Gaussian functions. Each time an ant walks through a continuous variable, corresponding PDF is sampled and so the part of a new solution candidate is generated. Other features of ACO like pheromone reinforcement and evaporation are retained.

The *Direct Ant Colony Optimization* (DACO) (Kong & Tian, 2006) uses two types of pheromones — one for mean values and one for standard deviations. These values are used by ants to create new solutions and are updated in the ACO way.

The *Adaptive Ant Colony Algorithm* (AACA) (Li & Wu, 2003) encodes solutions into binary strings. Ants travel from the most significant bit to the least significant bit, choosing nodes 0 or 1 for each step. After finishing the trip, the resulting binary string is converted into a solution candidate and the pheromone on the path is reinforced according to the quality of the solution. The probability of a change in more significant bits decreases during algorithm run.

The API algorithm (Venturini & Slimane, 2000) is named after *Pachycondyla apicalis* and it simulates the foraging behavior of these ants. Ant moves from a nest to a one of several hunting sites, generated in its neighborhood. It performs a random move in the small neighborhood of the selected hunting site to generate a new solution. If an improvement occurs, the next search leads to the same hunting site. If the hunt is unsuccessful more than *p* times for a single hunting site, the hunting site is forgotten and ant randomly generates a new one. After some time period the nest is moved to the best of all solutions.

3.2.5. *Hybrid search*

The *Hybrid of the GA and the PSO* (HGAPSO) algorithm was proposed in Juang and Liou (2004). PSO works based on a social adaptation of knowledge, and all individuals are considered to be of the same generation. On the contrary, GA is based on the evolution over successive generations, so the changes of individuals during a single generation are not considered. In nature, individuals will grow up and become more suitable to the environment before producing offspring. To incorporate this phenomenon into GA, PSO is adopted to enhance the top-ranking individuals of each generation.

3.2.6. *Other methods*

The *Orthogonal Search* (OS) optimizes a multivariate problem by selecting one dimension at a time, minimizing the error at each step. The OS, also known as the *Powell's method*, was used in Adeney and Korenberg (2000) to train single layered neural networks. In our experiments, we use two implementations designated the OS and the OS-pal (see Drummond and Strimmer (2001)) differing in a type of line search method and stopping criteria. The *Stochastic Orthogonal Search* (SOS) differs from OS just by random permutations of dimension order.

As the last method we have employed a simple Random search (RANDOM), where in each iteration the most fit solution out of multiple randomly generated is kept.

3.3. *Visual inspection of the optimization process*

Based on experiments with different types of neurons in the network performed on various data sets we propose the following way of network optimization visual inspection process.

3.3.1. *Starting point*

The starting point is an initial configuration of weights for the iterative method to start with. For example in case of Sigmoid neuron, it is sufficient to supply a random starting point around zero (weights within (−0.3, 0.3) for normalized data). Large initial weighs prevent optimization method to converge, because the sigmoid function is saturated (it is sensitive just in almost linear part around zero).

$$y_j = \mathrm{e}^{-\frac{\sum_{i=1}^{n}(x_{ij}-a_i)^2}{2*(a_{n+i})^2}}. \tag{11}$$

In case or Gaussian neuron (Eq. (11)), weights a_{n+i} should not be too small. Then, the error surface is flat with a single deep pit and the optimization method is unable to locate this pit with a global minima.

For Gaussian neuron, we use the maximum likelihood estimate of weights (means and standard deviations computed on training data) as the starting point.

3.3.2. *Error surface inspection*

We have implemented an inspection tool allowing us to monitor the the optimization process. The training error is dependent variable and weights are independent variables. To be able to see, how the training error surface looks like, we need to use projections of the multidimensional space. We use scatterplot matrix of training error plots. For each plot, two weights are varied in the interval (−15, 15), all other weights are fixed (values in actual iteration) and the training error is computed in each point of the plot. The darker the background is, the lower the training error. Iteration history in each plot is visualized in order to observe the process convergence.

This visualization is particularly useful for getting information on training error surface complexity in each dimension (how a change of individual weight influences the error).

In Fig. 5, you can observe the training log of the single sine neuron on the Bosthouse data set (Kordík, 2006b). The Quasi-Newton method (QN) needed almost 1400 iteration to find optimal values of 24 weights (Bosthouse has 12 features, the output of the neuron is $y = \sum_{i=1}^{12} \sin(a_i * x_i + a_{12+i})$). The algorithm managed to escape from a local minima (iteration 170) and converged to a global optima.

3.4. *Analytic gradient*

Note that supplying the optimization method the the analytic gradient is not always the best option. In the optimization of the Sine neuron (Fig. 5), the gradient was not supplied and QN method estimated it numerically. Therefore much more iterations was needed. On the other hand, when we supply the analytic gradient, QN relies on it much more and get stuck in a local minima after 45 iterations.

In deeper layers of the network, error surface of units becomes increasingly complex and the analytical gradient is important, because it saves large portion of the computational time (see Table 1).

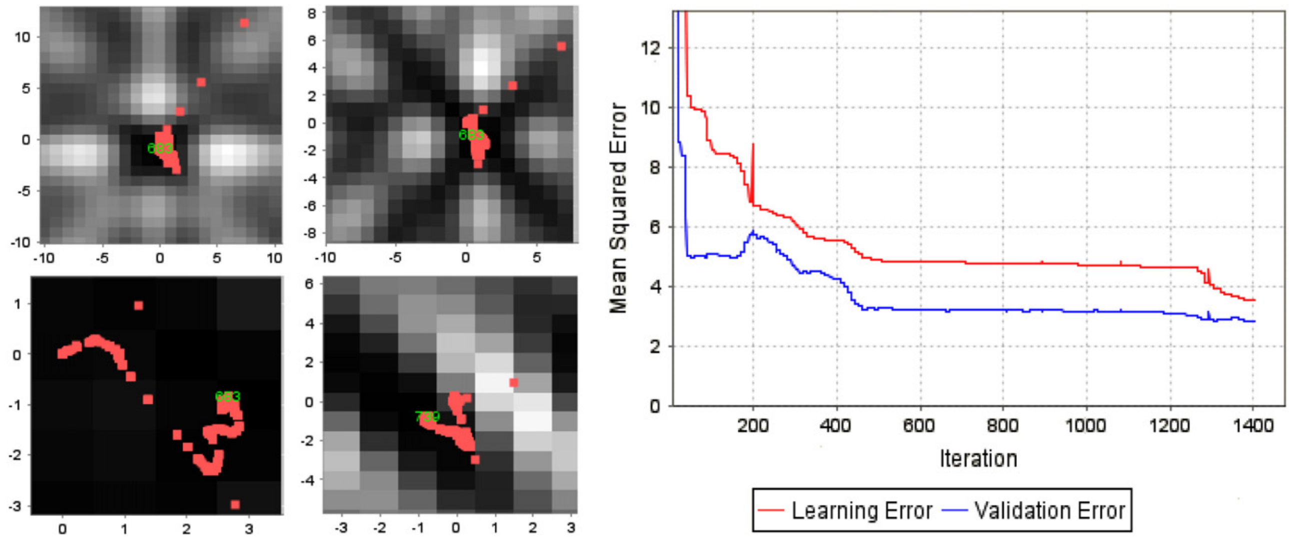


Fig. 5. The Sine neuron optimized by Quasi-Newton algorithm with numerical estimates of the analytic gradient. Visualization of the training error surface from different perspectives (left) and the convergence history (right).

4. Optimization method benchmarks

In this section, we evaluate a performance of individual optimization algorithms. Each algorithm optimizes coefficients of all neurons within network build on a benchmarking data set.

4.1. *Tested data sets*

In our experiments we utilize a well known collection of data sets — the **Proben1** (Prechelt, 1994). Among data sets in Proben1 collection we selected following data sets:

- Cancer — the problem is to discriminate between belign and malign tumors of breast cancer.
- Diabetes — predicts a diabetes in Pima Indians. Input variables are e.g. age, number of children and outcome of medical examinations.
- Gene — the task is to identify intron and exon boundaries in nucleotic sequences.
- Glass — the task is to identify original purpose of glass fragments based on their chemical composition.
- Heart — predicts heart disease. The outcome is if at lease one of four main vessels is reduced by more that 50%. This data set is union of four source data sets.
- Heartc — the same as previous data set, but contains only one source data sets.
- Horse — the task is to decide on fate of a horse which has a colic. Input attributes are based on veterinary examination.

Data sets Cancer, Diabetes, Gene and Glass originates in UCI Machine Learning repository. Data sets in Proben1 are divided into training, testing and validation sets. Since in our experiments we utilize a cross-validation approach we can merge all three parts into one data set and all experiments works with this single set.

Apart from data sets from Proben1, we use Two Intertwined Spirals data set (Koza, 1992) and also two unknown data sets described below.

4.1.1. *Mandarin data set*

The Mandarin tree data set describes water consumption of a mandarin tree. The mandarin tree is the complex system influenced by many input variables (water, temperature, sunshine, humidity of the air, etc.).

Our data set consists of measurements of these input variables and one output variable — the water consumption of the tree. It describes how much water the tree needs in specific conditions. We used 2500 training vectors (11 input variables, 1 output variable).

4.1.2. *Antro data set*

represents a set of observations. the skeletal indicators studied for the proposal of the methods of age at death assessment from the human skeleton (see Gambier, Bruzek, Schmitt, Houet, and Murail (2006)). It is a results of the visual scoring of the morphological changes of the features in two pelvic joint surfaces defined and described by a text accompanied with photos. The material consists of 955 subjects from the 9 human skeletal series of subjects known age and sex. This collections (populations) are dispersed on 4 continents (Europe, North America, Africa, Asia). The age in the death of the individuals varies between 19 and 100 years.

4.2. *Configuration of experiments*

To test our optimization methods we will perform experiments with three different configurations. In all three configurations other parameters of training algorithm will remain constant and results will measure only an influence of the optimization method.

- *A single optimization method enabled* — in this configuration only one optimization method is enabled and resulting models will be created using only this method. The goal is to evaluate performance of each optimization algorithm for given data set.
- *All optimization methods enabled* — in this configuration all optimization methods are enabled. Each neuron in GAME model may be optimized by different method. The goal is to test the meta-optimization algorithm implemented in GAME training algorithm.
- *Best optimization methods enabled* — this configuration is almost the same as previous one and with the same goal. Only difference is that only two optimization methods are enabled. We choose CMA-ES and Quasi-Newton optimization methods which gave the best results in the first experiment.

All results will be produced using cross-fold validation scheme with 10 folds. To improve reliability of our experiments we repeated each cross-fold experiment 10 times. This will produce 100 models which should be enough to obtain statistically significant results.

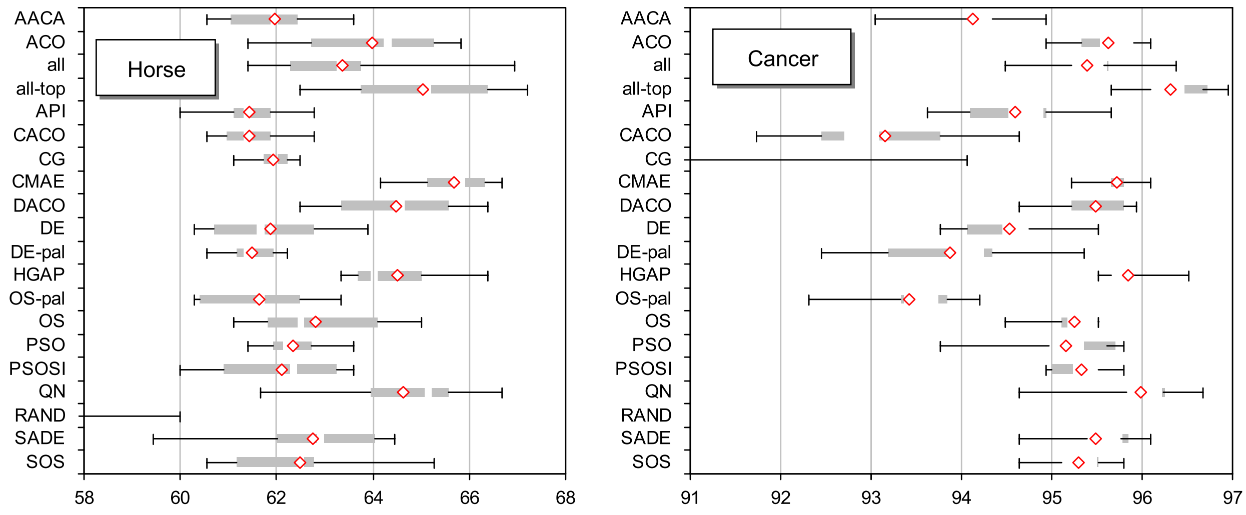


Fig. 6. Box plots showing classification accuracy of networks optimized by individual algorithms on Horse and Cancer data sets. Box plots of individual optimization algorithms were combined into a single graph for each data set.

4.3. Results

Fig. 6 shows performances of individual algorithms on Horse and Cancer from the Proben1 and UCI repositories. Algorithms "all" and "all-top" will be explained in the next section. For the Horse data set, the CMA-ES algorithm performed significantly better than all other individual algorithms. For the Cancer data set, the QN and HGAPSO algorithms built most accurate networks.

Note that DE and DE-pal are two versions of the Differential Evolution algorithm implemented in the GAME engine. Also OS and OS-pal represent the same algorithm with slightly different configuration (e.g. different type of line search). The first versions of methods often outperformed the "pal" versions. It signifies that a proper configuration of algorithms is very important.

For the Glass data set (Fig. 7 left), the best algorithm was the QN followed by CMA-ES and DACO. On the Two intertwined spirals problem (Fig. 7 right) results were completely different. The only optimization algorithms capable of training successful networks were Ant colony based DACO and AACA. All other algorithms failed to produce useful classifiers.

The surprising result was analyzed and found out that the final network is composed of neurons with Sine, Sigmoid and Gaussian transfer function. The ACO algorithm is capable to find underlying relationship by complex combination of periodic sine functions. Other methods were sometimes successful on the training data only, more often they failed to classify even the training data.

5. Combination of optimization methods

In this section we explain the *all* and *all-top* algorithm, that can be found in Figs. 6 and 7. We assumed that for each data set, some optimization methods are more efficient than others. If we select an appropriate method to optimize coefficients of each neuron within a single GAME network, the accuracy will increase. The problem is to find out which method is the appropriate (and the most effective).

In the *all* optimization algorithm, we used a simple strategy. When a new neuron was generated, a random method was assigned to optimize the coefficients of the neurons. In case the optimization method was inappropriate, coefficients were not set optimally and neuron did not survive in the genetic algorithm evolving neurons in a layer of the GAME network. Only the appropriate optimization methods were able to generate the fittest neurons (Fig. 8).

The remarkable point of this approach is that optimization methods are combined within a single model. Such combination is widely used in meta-learning and resembles the Cascade generalization strategy. For example a sigmoidal neuron optimized by the QN algorithm simplifies the problem for a sine neuron optimized by the DACO algorithm and also for an output of a linear neuron which is connected to both sigmoidal and sine neurons. Another example of network optimized by more algorithms is presented in Fig. 9.

Let us discuss the results of proposed meta-optimization methods *all* and *all-top*.

The *all* configuration combines all optimization methods from the Table 2. When you look at Fig. 7, the accuracy of networks produced by *all* optimization methods combined strategy is above average. Note that the maximal accuracy is higher than that of any individual optimization method. However the standard deviation of results is very high. This is caused by insufficient size of population in the niching genetic algorithm. Default number of neurons in the population was 15 and maximal number of generations 30. Therefore some (possibly fittest) optimization methods were not even used once in the population of neurons. And there are more assumptions that have to be met for optimization method to produce fit neuron — the neuron must have good input connections and good transfer function.

Increasing the population size (and computational complexity at the same time) is the obvious solution that improves results of the *all* strategy.

The next solution possible is to combine just a few optimization methods that exhibit superb and stable results on larger group of data sets. We have selected the best performers: the QN and the CMA-ES methods only and created the *all-top* configuration. The advantage of this strategy is that it combines best gradient based with best evolutionary based method. In Fig. 6 one can see that although number of neurons in the population stays small, the accuracy significantly improved. For the Cancer data set, *all-top* is superior to all other methods.

In Fig. 7 one can see results consistent with previous conclusions. The only interesting deviation is that the *all* scores better than the *all-top* on the Spiral data. This can be easily explained — successful colony methods AACA and DACO are not included in the *all-top* configuration.

In future, we plan to repeat these experiments with increased population size. Our impression is that the *all* configuration has potential to become best performing optimization strategy — when

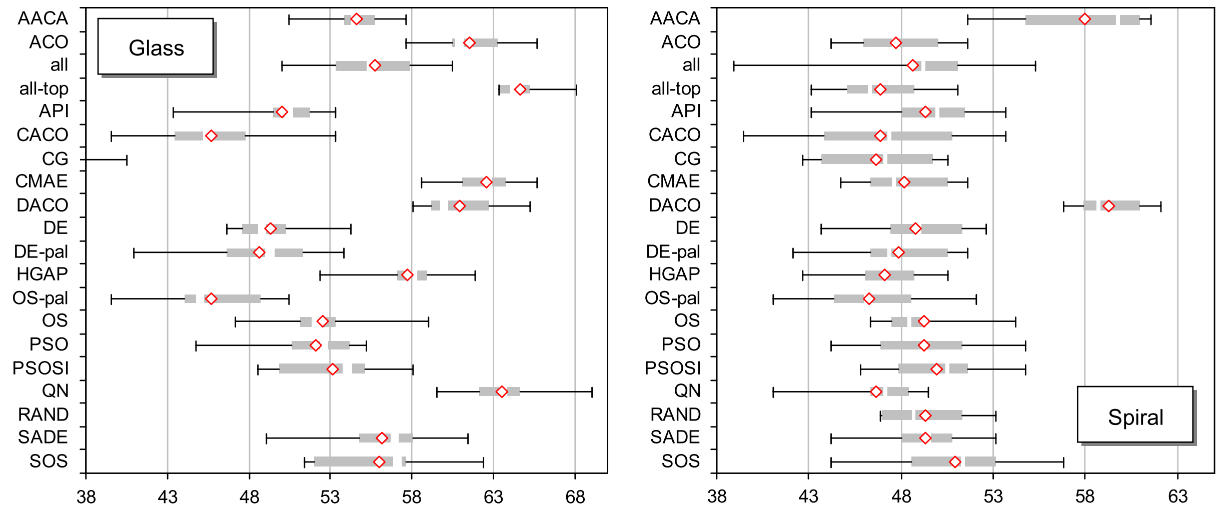
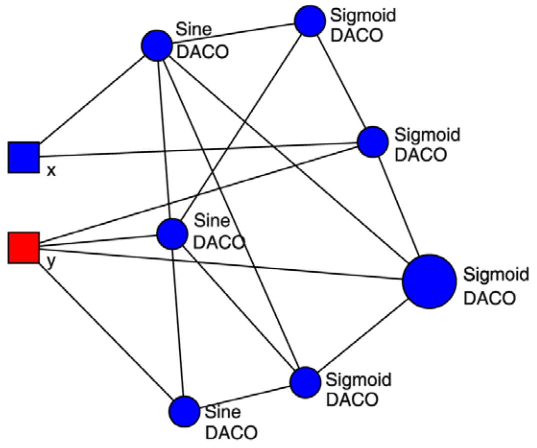


Fig. 7. Box plots showing classification accuracy of networks optimized by individual algorithms on Glass and Spiral data sets.



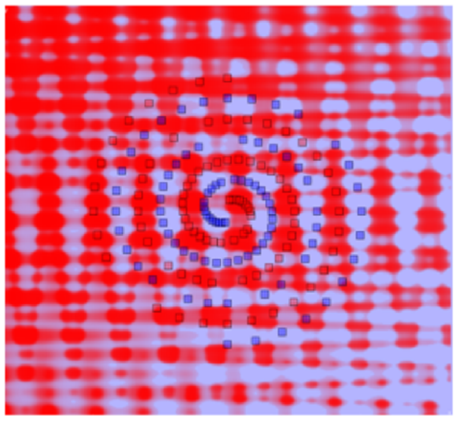


Fig. 8. The structure of single GAME network build on the Spiral data, neurons optimized by the DACO algorithm (left). The output encoded by color of background. Rectangles are training vectors. The same color of rectangle and background signifies well trained vectors (right).

the search space of possible neuron connections, transfer functions and optimization methods is sufficiently explored.

To increase the efficiency of the search for successful optimization methods, we propose an evolutionary approach to replace the random search employed above.

5.1. *Inheritance of optimization methods*

Parent neurons have often inputs and a transfer function similar to their children. The type of optimization method can be inherited from parents, because neurons are evolved by means of niching genetic algorithm (NGA). This evolutionary algorithm can also assign appropriate optimization methods to neurons being evolved. We added a type of an optimization method into the chromosome of neurons. When new neurons are generated by crossover to the next generation, they also inherit a type of optimization method from their parent neurons. The expected outcome is that methods, training successful neurons, are selected more often than methods, training poor performers on a particular data set.

Again, an experiment was designed to prove this assumption.

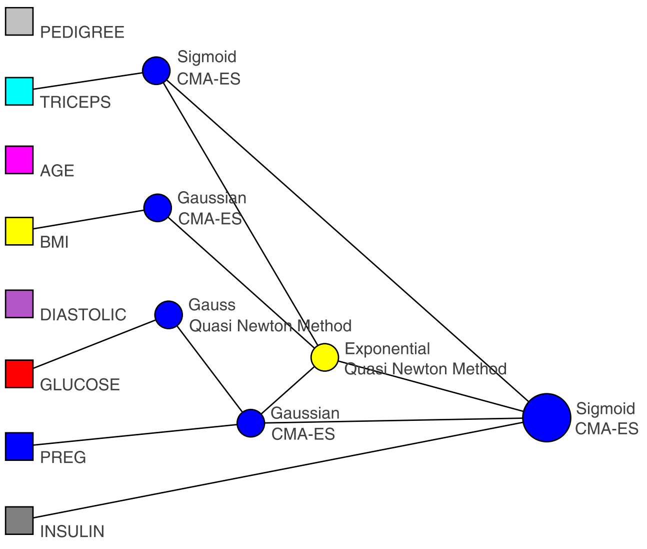


Fig. 9. One GAME network developed by all-top configuration on Diabetes data set.

5.1.1. *The experiment*

We prepared configurations of the GAME engine with several different inheritance settings. In the configuration *p*0% new neurons inherit an optimization method from their parent neurons. In the configuration *p*50% offspring has a 50% chance to get random method assigned. In the configuration *p*100% nothing is inherited, all optimization methods are set randomly.

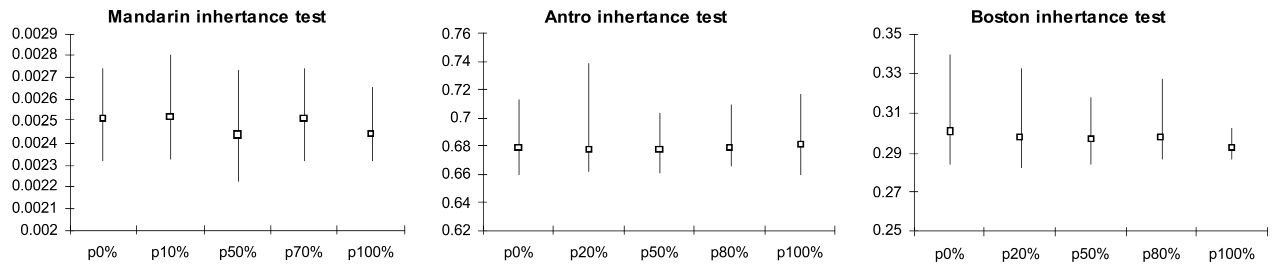


Fig. 10. Experiments with the inheritance of transfer function and learning method. Probability of random settings replacing inherited settings is shown. For all three data sets, the fifty percent inheritance probability is a reasonable choice, but results are not very significant.

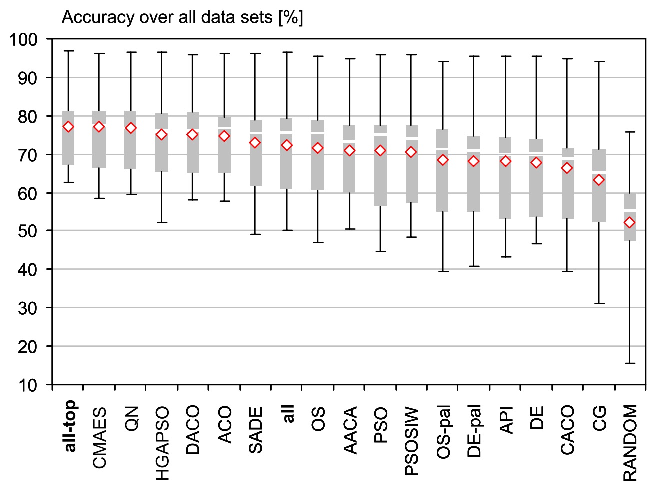


Fig. 11. The accuracy of methods over all data sets. The *all-top* is a meta-optimization method combining QN with CMAES.

We have been experimenting with the Mandarin, Antro and Boston data sets. For each configuration 30 networks were evolved. The maximum, minimum and mean of their RMS errors for each configuration are displayed in Fig. 10. Results are very similar for all configurations and data sets. There is no configuration significantly better than others. For all data sets we can observe that the *p*50% and the *p*100% configuration have slightly better mean error values and lower dispersion of errors. We chose the *p*50% configuration to be default in the GAME engine. It means offspring neurons have a 50% chance to get random optimization method assigned otherwise their methods are inherited from parent neurons.

Again, results will be more significant, when bigger population of neurons in more generations of NGA will be utilized in experiments.

Finally, we would like to answer the question which optimization method performed best in our experiments.

6. Evaluation of methods over more data sets

Fig. 11 averages the results over following group of data sets: Cancer, Diabetes, Gene, Glass, Heart, Heartc, Horse. The final box plot of accuracy for each method was computed from all results of ten fold cross validation over all data sets (70 numbers for each method).

The *all-top* meta-optimization approach produces most accurate classifiers and also the deviation in their accuracy is the lowest. As you can see, the performance of best three methods *all-top*, *CMA-ES* and *QN* is very similar. When speed of methods should be considered, the fastest method is QN (thanks to analytic gradients), *CMA-ES* was in average 70% slower and *all-top* is in between the two methods.

Note that for some data sets, *all-top* produces classifiers with highest accuracy so far (see Fig. 6). Here the combination of different optimization methods within one network and the diversity of the methods bring higher quality — it is superior to networks optimized by single methods. The meta-optimization strategy is very useful notably in case, where the aim is to use an optimization strategy performing well on many different data sets.

7. Conclusion

We have shown that core principles of meta-learning such as diversity promotion, specialization or weighted combination of weak learners can be beneficial for neural network optimization task.

We presented efficient optimization strategy for hybrid neural networks based on the principles above.

We made a comprehensive overview of techniques used to optimize individual neurons of neural networks. The benchmark of optimization algorithms on several data sets showed that the CMA-ES and the Quasi-Newton method performed consistently very well on majority of data sets. The QN method is often faster because it can utilize analytic gradients derived for neurons in our neural network.

We also proposed a meta-optimization strategy combining these two methods in order to get consistently good results on wide range of data sets. Our strategy outperformed all individual methods on some data sets and performs best over all data sets tested.

The evolution of optimization methods is very promising. Our future work is to employ meta-optimization approach in structural optimization instead of deterministic crowding method. We would like to test more advanced combination strategy to be able to benefit from slower but diverse optimization methods.

Acknowledgements

This research is partially supported by the grant Automated Knowledge Extraction (KJB201210701) of the Grant Agency of the Academy of Science of the Czech Republic and the research program "Transdisciplinary Research in the Area of Biomedical Engineering II" (MSM6840770012) sponsored by the Ministry of Education, Youth and Sports of the Czech Republic. The authors would like to thank Dr. Phil Prendergast from Hort Research, New Zealand who provided us with the Mandarin data set and Dr. Jaroslav Bruzek who provided us with the Antro data set and with many valuable comments of our results.

References

Adeney, K., & Korenberg, M. (2000). An easily calculated bound on condition for orthogonal algorithms. In *IEEE-INNS-ENNS international joint conference on neural networks* (p. 3620). *Vol. 3*.
Aine, S., Chakrabarti, P. P., & Kumar, R. (1992). An automated meta-level control framework for optimizing the quality-time tradeoff of VLSI algorithms. *IEEE Transactions on CAD of Integrated Circuits and Systems*, *26*(11), 2008, 2007.
Alpaydin, E., & Kaynak, C. (1998). Cascading classifiers. *Kybernetika*, *34*, 369–374.

Angeline, P. J., Saunders, G. M., & Pollack, J. P. (1994). An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, *5*(1), 54–65.

Bilchev, G., & Parmee, I. C. (1995). The ant colony metaphor for searching continuous design spaces. In *Selected papers from AISB workshop on evolutionary computing* (pp. 25–39). London, UK: Springer-Verlag.

Blum, C., & Socha, K. (2005). Training feed-forward neural networks with ant colony optimization: An application to pattern classification. In *Proceedings of hybrid intelligent systems conference* (pp. 233–238). Los Alamitos, CA, USA: IEEE Computer Society.

Boyan, J. (1998). *Learning evaluation functions for global optimization. Ph.D. thesis*. Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Brazdil, P., Giraud-Carrier, C., Soares, C., & Vilalta, R. (2009). *Cognitive technologies, Metalearning, applications to data mining*. Berlin, Heidelberg: Springer.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*(2), 123–140.

Brown, G. (2004). *Diversity in neural network ensembles. Ph.D. thesis*. The University of Birmingham, School of Computer Science, Birmingham B15 2TT, United Kingdom.

Buk, Z., Koutník, J., & Šnorek, M. (2009). NEAT in HyperNEAT substituted with genetic programming. In *LNCS: Vol. 5495. International conference on adaptive and natural computing algorithms*. Berlin, Heidelberg: Springer-Verlag.

Burke, E., & Kendall, G. (2005). *Search methodologies: Introductory tutorials in optimization and decision support techniques*. Springer.

Burke, E. K., Petrovic, S., & Qu, R. (2006). Case-based heuristic selection for timetabling problems. *Journal of Scheduling*, *9*(2), 115–132.

Cicirello, V. A. (2003). *Boosting stochastic problem solvers through online self-analysis of performance. Ph.D. thesis*. Carnegie Mellon University, Pittsburgh, PA, USA, 2003. Adviser-Smith, Stephen F.

Cramer, N. L. (1985). A representation for the adaptive generation of simple sequential programs. In *Proceedings of the 1st international conference on genetic algorithms* (pp. 183–187). Hillsdale, NJ, USA: L. Erlbaum Associates Inc.

D'Ambrosio, D. B., & Stanley, K. O. (2007). A novel generative encoding for exploiting neural network sensor and output geometry. In *GECCO'07: Proceedings of the 9th annual conference on genetic and evolutionary computation* (pp. 974–981). New York, NY, USA: ACM.

D'Ambrosio, D. B., & Stanley, K. O. (2008). Generative encoding for multiagent learning. In *GECCO'08: Proceedings of the 10th annual conference on genetic and evolutionary computation* (pp. 819–826). New York, NY, USA: ACM.

Dennis, J. E., Jr., & Schnabel, R. B. (1996). *Numerical methods for unconstrained optimization and nonlinear equations (classics in applied mathematics, 16)*. Society for Industrial and Applied Mathematics.

de Oca, M. A. M., Stützle, T., Birattari, M., & Dorigo, M. (2006). A comparison of particle swarm optimization algorithms based on run-length distributions. In *ANTS workshop* (pp. 1–12).

Deugo, D., & Ferguson, D. (2004). Evolution to the xtreme: Evolving evolutionary strategies using a meta-level approach. In *Proceedings of the 2004 IEEE congress on evolutionary computation* (pp. 31–38). Portland, Oregon: IEEE Press.

Diosan, L. S., & Oltean, M. (2007). Evolving evolutionary algorithms using evolutionary algorithms. In *GECCO'07: Proceedings of the 2007 GECCO conference companion on genetic and evolutionary computation* (pp. 2442–2449). New York, NY, USA: ACM.

Drchal, J., Kučerová, A., & Němeček, J. (2002). Optimizing synaptic weights of neural networks. In *ICECT'03: Proceedings of the third international conference on engineering computational technology* (pp. 211–212). Edinburgh, UK: Civil-Comp press.

Drummond, A., & Strimmer, K. (2001). Pal: An object-oriented programming library for molecular evolution and phylogenetics. *Bioinformatics*, *17*(7), 662–663.

Eiben, A. E., Michalewicz, Z., Schoenauer, M., & Smith, J. E. (2007). Parameter control in evolutionary algorithms. In *Parameter setting in evolutionary algorithms* (pp. 19–46). Springer.

Engelbrecht, A. (2002). *Computational intelligence: An introduction*. New York, NY, USA: Halsted Press.

Fahlman, S. E. (1988). An empirical study of learning speed in back-propagation networks. *Technical report CMU-CS-88-162*. Carnegie Mellon Univ.

Fahlman, S., & Lebiere, C. (1991). The cascade-correlation learning architecture. In *Advances in neural information processing systems 2* (pp. 524–532). Morgan Kaufmann.

Ferri, C., Flach, P., & Hernández-Orallo, J. (2004). Delegating classifiers. In *ICML'04: Proceedings of the twenty-first international conference on machine learning* (p. 37). New York, NY, USA: ACM.

Fogel, D., Fogel, L., & Atmar, J. (1991). Meta-evolutionary programming. In *Signals, systems and computers, 1991. 1991 Conference record of the twenty-fifth asilomar conference on* (pp. 540–545). *Vol. 1*.

Freund, Y., & Schapire, R. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the second European conference on computational learning theory* (pp. 23–37). Springer Verlag.

Gama, J., & Brazdil, P. (2000). Cascade generalization. *Machine Learning*, *41*(3), 315–343.

Gambier, D., Bruzek, J., Schmitt, A., Houet, F., & Murail, P. (2006). Age at death and sex diagnosis of the cro-magnon fossils (Dordogne, France) based on the pelvic bone. *Comptes Rendus Palevol*, *5*, 735–741.

Gauci, J., & Stanley, K. (2007). Generating large-scale neural networks through discovering geometric regularities. In *GECCO'07: Proceedings of the 9th annual conference on genetic and evolutionary computation* (pp. 997–1004). New York, NY, USA: ACM.

Glover, F. W., & Kochenberger, G. A. (2003). *Handbook of metaheuristics (international series in operations research & management science)*. Springer.

Gomez, F., & Miikkulainen, R. (1996). Incremental evolution of complex general behavior. *Adaptive Behavior*, *5*, 5–317.

Gomez, F. J., & Schmidhuber, J. (2005). Co-evolving recurrent neurons learn deep memory pomdps. In *GECCO'05: Proceedings of the 2005 conference on genetic and evolutionary computation* (pp. 491–498). New York, NY, USA: ACM.

Gomez, F., Schmidhuber, J., & Miikkulainen, R. (2008). Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research*, *9*, 937–965.

Grefenstette, J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on System Man, and Cybernetics*, *16*(1), 122–128.

Gruau, F. (1994). *Neural network synthesis using cellular encoding and the genetic algorithm. Ph.D. thesis*. Ecole Normale Superieure de Lyon, France.

Hansen, N., & Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, *9*(2), 159–195.

Hart, W., Krasnogor, N., & Smith, J. (Eds.). (2004). *Recent advances in memetic algorithms*. Berlin, Heidelberg, New York: Springer.

Holland, J. (1975). *Adaptation in neural and artificial systems*. University of Michigan Press.

Ho, Y., & Pepyne, D. (2002). Simple explanation of the no-free-lunch theorem and its implications. *Journal of Optimization Theory and Applications*, *115*, 549.

Hrstka, O., & Kučerová, A. (2004). Improvements of real coded genetic algorithms based on differential operators preventing premature convergence. *Advances in Engineering Software*, *35*(3–4), 237–246.

Ivakhnenko, A. G. (1971). Polynomial theory of complex systems. *IEEE Transactions on Systems, Man, and Cybernetics*, *SMC-1*(1), 364–378.

Juang, C.-F., & Liou, Y.-C. (2004). On the hybrid of genetic algorithm and particle swarm optimization for evolving recurrent neural network. In *Proceedings of the IEEE international joint conference on neural networks* (pp. 2285–2289) *Vol. 3*. Dept. of Electr. Eng., Nat. Chung-Hsing Univ., Taichung, Taiwan, 25–29.

Kaynak, C., & Alpaydin, E. (2000). Multistage cascading of multiple classifiers: One man's noise is another man's data. In *ICML'00: Proceedings of the seventeenth international conference on machine learning* (pp. 455–462). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Kong, M., & Tian, P. (2006). A direct application of ant colony optimization to function optimization problem in continuous domain. In *ANTS workshop* (pp. 324–331).

Kordík, P. (2006a). *Fully automated knowledge extraction using group of adaptive models evolution. Ph.D. thesis*. Czech Technical University in Prague, FEE, Dep. of Comp. Sci. and Computers, FEE, CTU Prague, Czech Republic.

Kordík, P. (2006b) *Fully automated knowledge extraction using group of adaptive models evolution. Ph.D. thesis*. Czech Technical University in Prague, Praha.

Kordík, P. (2009). *Studies in computational intelligence*: *Vol. 211. Hybrid self-organizing modeling systems* (p. 290). Berlin, Heidelberg: Springer-Verlag.

Kordík, P., Kovářík, O., & Šnorek, M. (2007). Optimization of models: Looking for the best strategy. In *Proceedings of the 6th EUROSIM congress on modelling and simulation: Vol. 2* (pp. 314–320). Vienna: ARGESIM.

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA, USA: MIT Press.

Li, Y.-j., & Wu, T.-j. (2003). An adaptive ant colony system algorithm for continuous-space optimization problems. *Journal of Zhejiang University Science*, *4*(1), 40–46.

Ivakhnenko, A. G., & Müller, J.-A. (1995). Self-organization of nets of active neurons. *System Analysis Modelling and Simulation*, *20*(1–2), 93–106.

Mahfoud, S. W. (1995). Niching methods for genetic algorithms. *Technical report 95001*. Illinois Genetic Algorithms Laboratory (IlliGaL), University of Ilinios at Urbana-Champaign.

Mahfoud, S. W. (1995). A comparison of parallel and sequential niching methods. In *Sixth international conference on genetic algorithms* (pp. 136–143).

Meissner, M., Schmuker, M., & Schneider, G. (2006). Optimized Particle Swarm Optimization (OPSO) and its application to artificial neural network training. *BMC Bioinformatics*, *7*, 125.

Merz, P., & Freisleben, B. (2000). Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation*, *4*(4), 337–352.

Moriarty, D. E., & Miikkulainen, R. (1997). Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation*, *5*(4), 373–399.

Muller, J. A., & Lemke, F. (2000). *Self-organising data mining*. Berlin.

Murre, J. M. J. (1992). *Learning and categorization in modular neural networks*. Harvester Wheatsheaf.

Ong, Y.-S., & Keane, A. J. (2004). Meta-lamarckian learning in memetic algorithms. *IEEE Transactions on Evolutionary Computation*, *8*(2), 99–110.

Onwubolu, G. C., & Davendra, D. (2009). *Differential evolution: A handbook for global permutation-based combinatorial optimization*. Springer Publishing Company, Incorporated.

Ortega, J. (1996). *Making the most of what youve got: Using models and data to improve prediction accuracy. Ph.D. thesis*. Vanderbilt University.

Ortega, J., Koppel, M., & Argamon, S. (2001). Arbitrating among competing classifiers using learned referees. *Knowledge and Information System*, *3*(4), 470–490.

Poli, R., & Graff, M. (2009). Free lunches for neural network search. In *GECCO'09: Proceedings of the 11th annual conference on genetic and evolutionary computation* (pp. 1291–1298). New York, NY, USA: ACM.

Poli, R., Graff, M., & McPhee, N. F. (2009). Free lunches for function and program induction. In *FOGA'09: Proceedings of the tenth ACM SIGEVO workshop on foundations of genetic algorithms* (pp. 183–194). New York, NY, USA: ACM.

Prechelt, L. (1994). PROBEN1 — A set of benchmarks and benchmarking rules for neural network training algorithms. *Technical report 21/94*. Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany.

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution*. Frommann-Holzboog.

Riedmiller, M., & Braun, H. (1992). Rprop — A fast adaptive learning algorithm. *Technical report*. Universitat Karlsruhe.

Ritchie, M. D., White, B. C., Parker, J. S., Hahn, L. W., & Moore, J. H. (2003). Optimization of neural network architecture using genetic programming improves detection and modeling of gene–gene interactions in studies of human diseases. *BMC Bioinformatics*, *4*(1), 28+.

Roli, A., & Milano, M. (2004). MAGMA: A multiagent architecture for metaheuristics. *IEEE Transactions on Systems, Man and Cybernetics — Part B*, *34*, 2002.

Rosenblatt, F. (1957). The perceptron: A perceiving and recognizing automaton. *Technical report 85-460-1*.

Salane, & Tewarson (1980). A unified derivation of symmetric quasi-newton update formulas. *Applied Math*, *25*, 29–36.

Satoru Hiwa, T. H., & Miki, M. (2007). Hybrid optimization using direct, ga, and sqp for global exploration. In *IEEE Proceedings of 2007 congress on evolutionary computation* (pp. 1709–1716).

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, *5*(2), 197–227.

Schmidhuber, J. (1997). Discovering neural nets with low Kolmogorov complexity and high generalization capability. *Neural Networks*, *10*(5), 857–873.

Schmidhuber, J. (1987). Evolutionary principles in self-referential learning. on learning now to learn: The meta-meta-meta...-hook. *Diploma thesis*. Technische Universitat Munchen, Germany.

Shewchuk, J. R. (1994). An introduction to the conjugate gradient method without the agonizing pain. *Technical report*. School of Computer Science Carnegie Mellon University, Pittsburgh, PA 15213.

Smith, J., & Fogarty, T. (1997). Operator and parameter adaptation in genetic algorithms. *Soft Computing*, *1*(2), 81–87.

Stanley, K. O. (2007). Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines*, *8*(2), 131–162.

Stanley, K. O. (2004). *Efficient evolution of neural networks through complexification. Ph.D. thesis*.

Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation*, *10*, 99–127.

Storn, R., & Price, K. (1997). Differential evolution — A simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, *11*, 341–359.

Tsoukalas, L. H., & Uhrig, R. E. (1996). *Fuzzy and neural approaches in engineering*. New York, NY, USA: John Wiley & Sons, Inc.

Tsutsui, S., Pelikan, M., & Ghosh, A. (2005). Performance of aggregation pheromone system on unimodal and multimodal problems. In *The IEEE congress on evolutionary computation, 2005* (pp. 880–887). *Vol. 1* IEEE.

Venturini, G., & Slimane, M. (2000). On how pachycondyla apicalis ants suggest a new search algorithm. *Future Generation Computer Systems*, *16*, 937–946.

Vesterstrom, J., & Thomsen, R. (2004). A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Proceedings of the 2004 congress on evolutionary computation* (pp. 1980–1987). *Vol. 2*.

Wade, J. G. (2006). Convergence properties of the conjugate gradient method. Available at www-math.bgsu.edu/~gwade/tex_examples/example2.txt.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*, *5*, 241–259.

Wolpert, D., & Macready, W. (1997). No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, *1*(1), 67–82.

Yao, X., & Liu, Y. (1997). Epnet for chaotic time-series prediction. In *Selected papers from the first Asia-Pacific conference on simulated evolution and learning* (pp. 146–156). Springer-Verlag.

Yuan, B., & Gallagher, M. (2007). Combining meta-EAs and racing for difficult EA parameter tuning tasks. In F. Lobo, C. Lima, & Z. Michalewicz (Eds.), *Parameter setting in evolutionary algorithms* (pp. 121–142). Springer.