Grzegorz DUDEK

Czestochowa University of Technology, Department of Electrical Engineering

doi:10.15199/48.2017.04.17

Stochastic Optimization Algorithms for Learning GRNN Forecasting Model – Comparative Study

***Abstract**. This paper presents stochastic optimization algorithms for learning Generalized Regression Neural Network which is used as a pattern-based short-term load forecasting model. For adjustment of the model parameters four types of stochastic optimization methods are used: evolution strategies, differential evolution, particle swarm optimization and tournament searching. The learning effectiveness when using these four algorithms is compared on real power system load data.*

***Streszczenie.** W artykule zaprezentowano stochastyczne algorytmy uczenia sieci neuronowej regresji uogólnionej, która pełni funkcję modelu krótkoterminowego prognozowania obciążeń elektroenergetycznych. Do strojenia parametrów modelu użyto czterech metod optymalizacji stochastycznej: strategii ewolucyjnych, ewolucji różnicowej, optymalizacji rojem cząstek i przeszukiwania turniejowego. Efektywność tych metod w uczeniu sieci porównano w badaniach symulacyjnych przy użyciu rzeczywistych danych. (**Stochastyczne algorytmy optymalizacji do uczenia modelu prognostycznego opartego na sieci GRNN – badania porównawcze**).*

Keywords: stochastic optimization algorithms, Generalized Regression Neural Network, short-term load forecasting.
Słowa kluczowe: stochastyczne metody optymalizacji, sieć neuronowa regresji uogólnionej, prognozowanie krótkoterminowe obciążeń.

Introduction

The main problem in building forecasting models is estimation of their parameter values. This is optimization problem in which the model error and sometimes the model complexity is minimized. Selection of the model optimization or learning method is related to the parameter types and the properties of the objective function. Linear objective function does not cause major problems. The simplex method, universal for this case, leads to the global minimum. If an objective function is nonlinear but continuous and differentiable, such as in multilayer perceptron, which is widely used as a forecasting model, gradient-based methods can be used, which unfortunately are suboptimal. Gradientless methods (direct search) do not require the differentiability of the objective function, but are still suboptimal. Selection of discrete parameter values (e.g. number of neurons) is performed using exhaustive enumeration, if the search space size is low. For spaces of bigger sizes the problem can be decomposed or the size can be reduced (dynamic programming, branch and bound method).

In recent years nature-inspired stochastic optimization methods, such as evolutionary algorithms and particle swarm optimization, are developed for solving many hard engineering problems. They are used for any kind of optimization problems: combinatorial, continuous, mixed, with constraints, multi-objective, dynamic, etc. The main feature of these methods is global optimization property, which allows the searching process to escape from the local minimum trap. In this work several stochastic optimization algorithms are tested for learning the forecasting model based on Generalized Regression Neural Network.

GRNN forecasting model

Generalized Regression Neural Network (GRNN) proposed by Specht [1] is a kind of Radial Basis Function (RBF) memory-based neural network with a one pass learning algorithm and highly parallel structure. It provides smooth function approximation with sparse data in a multidimensional space. Fast learning and easy tuning are the greatest advantages of GRNN.

The GRNN architecture is shown in Fig. 1. It is composed of four layers: input, RBF, summation and output ones. Input data are nonlinearly transformed by radial basis activation functions which usually have the form of Gaussian functions:

$$G_i(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{c}_i\|^2}{s_i^2}\right) \tag{1}$$

where $\mathbf{c}_i$ is a center vector, s_i is a bandwidth and $\|.\|$ is a Euclidean norm.

Each RBF neuron represents one training pattern. Thus, the number of RBF neurons is equal to the number of training patterns N. The center of i-th neuron is $\mathbf{c}_i = \mathbf{x}_i$. The neuron output expresses the similarity between the input pattern $\mathbf{x}$ and the i-th training pattern. The RBF layer maps n-dimensional input space into N-dimensional space of similarity. Responses of RBF neurons play the role of weights of corresponding target patterns $\mathbf{y}_j$. The linear combination of the target patterns is the GRNN output:

$$g(\mathbf{x}) = \frac{\sum_{i=1}^{N} G_i(\mathbf{x})\mathbf{y}_i}{\sum_{i=1}^{N} G_i(\mathbf{x})} \tag{2}$$

An input pattern $\mathbf{x}_i = [x_{i,1}\ x_{i,2}\ \dots\ x_{i,n}]$ represents a daily sequence of the load time series: $\mathbf{L}_i = [L_{i,1}\ L_{i,2}\ \dots\ L_{i,n}]$, where i is the day number and n is 24 hours. This is normalized load vector $\mathbf{L}_i$:

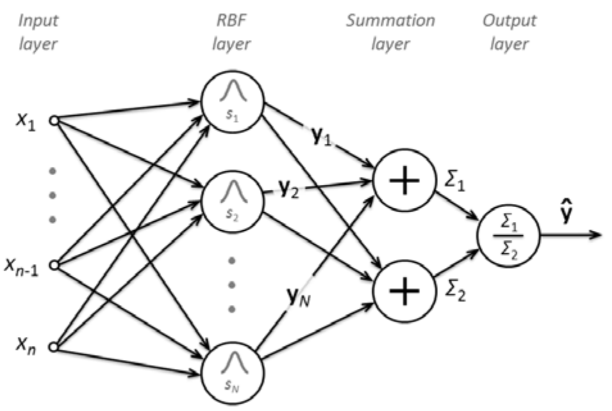

Fig.1. GRNN architecture

(3) $$x_{i,t} = \frac{L_{i,t} - \overline{L}_i}{\sqrt{\sum_{l=1}^{n}(L_{i,l} - \overline{L}_i)^2}}$$

where $t = 1, 2, ..., n$ and $\overline{L}_i$ is the mean load of the day i.

An output pattern $\mathbf{y}_i = [y_{i,1}\ y_{i,2}\ \dots\ y_{i,n}]$ represents a forecasted daily load sequence for the day $i+\tau$: $\mathbf{L}_{i+\tau} = [L_{i+\tau,1}\ L_{i+\tau,2}\ \dots\ L_{i+\tau,n}]$, where $\tau > 0$ is a forecast horizon:

(4) $$y_{i,t} = \frac{L_{i+\tau,t} - \overline{L}_i}{\sqrt{\sum_{l=1}^{n}(L_{i,l} - \overline{L}_i)^2}}$$

The goal of the time series representation by patterns is unification of data and filtering out annual and weekly variations as well as trend. This allows us to simplify the forecasting problem. More about time series patterns can be found in [2].

The GRNN model generates n-dimensional forecasted y-pattern as an output. To transform this vector into forecasted load vector $\mathbf{L}_{i+\tau}$ we use transformed equation (4).

The only GRNN parameters to be adjusted are bandwidths s_j governing the smoothness of the regression function (2). As bandwidth becomes larger the RBF neuron output increase (y-pattern weight in (2) increase), with the result that the fitted function becomes smoother. To find the optimal values for bandwidths stochastic optimization methods are applied described in the next section.

Stochastic optimization algorithms for GRNN learning

The solution $\mathbf{s} = [s_1, s_2, \dots, s_N]$ is searched to minimize the forecast error $e(\mathbf{s})$ estimated on the training set (as an error measure MAPE is used). For this continuous optimization problem of model fitting four stochastic methods are used that sample the search space using different heuristics.

Evolution strategies

In evolution strategies (ES) a population of individuals is processed. An individual in composed of three elements: a vector of variables $\mathbf{s}$, a vector of endogenous parameters $\boldsymbol{\sigma} = [\sigma_1, \sigma_2, \dots, \sigma_N]$ and an error function value at the point $\mathbf{s}$: $z = (\mathbf{s}, \boldsymbol{\sigma}, e(\mathbf{s}))$. The components of vector $\boldsymbol{\sigma}$ correspond to the components of the vector $\mathbf{s}$. They control the statistical properties of the mutation operator. The inclusion of algorithm parameters to the structure of an individual and their adaptation in evolutionary process, which takes place parallel to searching of variables, is a specific feature of ES.

ES denoted symbolically $(\mu/\rho+\lambda)$ [3] is used in this study. In one step (generation) of ES algorithm, λ offsprings are created from μ parents ($\lambda > \mu$). To create an offspring, ρ parent individuals are selected randomly from the population. Their features are combined during recombination using discrete crossover. It performs an exchange of $\mathbf{s}$ and $\boldsymbol{\sigma}$ components between the individuals. For each position the parent who contributes its component to the offspring is chosen randomly with equal probability:

(5) $$\sigma_i = \sigma_i^r, \quad s_i = s_i^r$$

where $i = 1, 2, ..., N$ and $r = \mathrm{rand}\{1, 2, \dots, \rho\}$ is an parent index.

The second genetic operator is mutation. In the first phase vectors $\boldsymbol{\sigma}$ of each offspring are mutated:

(6) $$\boldsymbol{\sigma}' = \exp(\zeta_0)[\sigma_1 \exp(\zeta), \sigma_2 \exp(\zeta), ..., \sigma_N \exp(\zeta)]$$

where: ζ_0 and ζ are random numbers drawn from normal distributions: $\zeta_0 \sim \mathrm{N}(0,(2N)^{-0.5})$ and $\zeta \sim \mathrm{N}(0,(2N^{0.5})^{-0.5})$.

In the next step vectors of variables $\mathbf{s}$ are mutated using new values of parameters σ'_i:

(7) $$\mathbf{s}' = [s_1 + \zeta_1, s_2 + \zeta_2, ..., s_N + \zeta_N]$$

where $\zeta_i \sim \mathrm{N}(0, \sigma'_i)$, $i = 1, 2, ..., N$.

While mutation of parameters σ_i consists in multiplying their values by random numbers from log-normal distribution, mutation of variables s_i consist in adding to their values random numbers from normal distribution with standard deviation σ'_i. Individuals learn correct values of parameters σ_i in an evolutionary process, adjusting them to the current state of the search process. In the initial stage of the search they take higher values, which will ensure increased exploration of the solution space. When the promising regions are identified, intensive exploration is no longer needed. The algorithm searches the space near the previously found solutions. This requires low values of the mutation parameters σ_i.

In the next step error function value $e(\mathbf{s})$ for each offspring is determined. Then population of parents and offsprings are combined and μ best individuals are selected. These individuals are parents in the next generation. These steps repeat until a stop criterion is met.

Differential evolution

Differential evolution (DE) is a heuristic algorithm for global optimization over continuous spaces [4], which is considered as the one of the most powerful stochastic optimization method [5]. In DE differences of the solution vectors are employed to explore the solution space. In our case solution vectors are $\mathbf{s}$. Populations of solution vectors (individuals) are processed in evolutionary process using mutation and crossover operators. Mutation consists of generating a mutant for each individual. A mutant for i-th individual is formed by adding a weighted difference between two randomly selected individuals to a third one:

(8) $$\mathbf{m}_i = \mathbf{s}_j + F(\mathbf{s}_k - \mathbf{s}_l)$$

where: $j, k, l \in \{1, 2, \dots, M\}\backslash i$ are randomly selected indexes (different from each other), M is a population size and $F \in [0, 2]$ is a coefficient controlling a mutation range.

The i-th individual is combined with a mutant generated for it using discrete crossover of the form:

(9) $$s'_{i,q} = \begin{cases} m_{i,q}, & \text{if } \xi_{i,q} \le CR \text{ or } q = \zeta_i \\ s_{i,q}, & \text{otherwise} \end{cases}$$

where: $q = 1, 2, \dots, N$, $\xi_{i,q}$ is a random number from uniform distribution $\mathrm{U}(0, 1)$, $\zeta_i = \mathrm{rand}\{1, 2, \dots, N\}$ is a randomly chosen index which ensures that the new solution gets at least one component of the mutant $\mathbf{m}_i$, and $CR \in [0, 1]$ is a crossover constant.

An offspring $\mathbf{s}'_i = [s'_{i,1}, s'_{i,2}, \dots, s'_{i,N}]$, created as a result of mutation and crossover, replaces its parent $\mathbf{s}_i$, if its error is lower than the parent error: $e(\mathbf{s}') < e(\mathbf{s})$. The mutation, crossover and replacement steps are repeated until a stop criterion is met.

Particle swarm optimization

Particle swarm optimization (PSO) is a stochastic, population-based, global optimization strategy that mimics social behavior of birds [6]. Individual birds in a flock (swarm) exchange information concerning their position, velocity and fitness. The flock behavior is influenced to increase the probability of migration to regions of high fitness.

PSO optimizes a problem using a population of candidate solutions (particles) and moving them around in the search space updating their positions and velocities. A particle movement depends on its velocity vector which is accelerated toward particle previous best position (where it had its lowest error value) and toward a swarm best position (the position of lowest error by any particle). In our problem the position of each particle is represented by the vector $\mathbf{s}$. New position of a particle is determined according to the formula:

(10) $$\mathbf{s}' = \mathbf{s} + \mathbf{v}'$$

where $\mathbf{v}'$ is a particle velocity updated according to the formula [7]:

(11) $$\mathbf{v}' = \chi[\mathbf{v} + c_1\boldsymbol{\zeta}_1 \otimes (\mathbf{s}^* - \mathbf{s}) + c_2\boldsymbol{\zeta}_2 \otimes (\mathbf{s}^{**} - \mathbf{s})]$$

where $\mathbf{s}^*$ is the position with the lower error found so far for the particle, $\mathbf{s}^{**}$ is the position with the lower error found so far for the swarm, χ is the constriction factor, c_1 and c_2 are the acceleration coefficients, $\boldsymbol{\zeta}_1$ and $\boldsymbol{\zeta}_2$ are two N-dimensional vectors of uniformly distributed random numbers independently generated within [0, 1], and $\otimes$ is the multiplication operator of the corresponding vector components.

Tournament searching

Tournament searching (TS) was designed for combinatorial optimization as an alternative to more complex stochastic algorithms such as genetic algorithm and simulated annealing [8]. It was expanded to continuous optimization problems and mixed binary-continuous problems (see [9], where TS was used for optimization of the forecasting model based on Nadaraya-Watson estimator).

Starting from the initial solution $\mathbf{s}$, TS explores the search space generating new trial solutions by perturbing the parent solution. In each iteration a set of L candidate solutions is generated from the parent solution using a move (or mutation) operator defined in the same manner as in ES (7), where $\zeta_i \sim \mathrm{N}(0, \sigma)$. The standard deviation σ controls the range of mutation. Its value decreases during the search process form a to 0 according to the formula:

(12) $$\sigma = a\left(1 - \frac{\ln(i)}{\ln(I_{\max})}\right)$$

where i is the iteration counter, $I_{\max}$ is the total number of iterations, $a > 0$ is a constant selected experimentally depending on values of variables s_i.

In the next step L candidate solutions are evaluated and the best one among them is selected (one with the lowest error $e(\mathbf{s})$). It becomes the parent solution in the next iteration.

The L parameter (called the tournament size) and the standard deviation σ control exploration/exploitation properties of the algorithm. For large L local minima attracts the searching process more intensively. For smaller L the probability of escaping from the attraction basin of local minima increases. But in this case the searching process is more random. The standard deviation σ determines the length of jumps, i.e. the distance between the parent solution and the candidate solutions generated from it.

Experimental evaluation

In this section, the proposed stochastic optimization algorithms for GRNN learning are evaluated experimentally in a number of forecasting tasks. By a forecasting task we mean: learn the GRNN model for forecasting the Polish power system load curve for the next day ($\tau = 1$). The test set for which the models are learned contains 61 days: 30 days from January 2004 (without atypical 1 January) and 31 days from July 2004. The training samples are selected individually for each forecasting task from the period from 1 January 2002 to the day preceding the forecasted day. They include pairs of patterns ($\mathbf{x}_i$, $\mathbf{y}_i$), where $\mathbf{y}_i$ represents the same day of the week (Monday, ..., Sunday) as the forecasted day.

Settings of the algorithms were as follows:

ES: $\mu = 30$, $\rho = 2$ i $\lambda = 7\mu$. Initial values of parameters σ_i was set to $0.06d_5$, where d_5 is the mean Euclidean distance between patterns $\mathbf{x}$ and their 5-th nearest neighbors in the training set.

DE: $CR = 0.1$, $F = 0.1$ (values recommended in [9], where DE was examined on the same forecasting tasks),

PSO: $\chi = 0.729$, $c_1 = c_2 = 2.05$ (values recommended in [7]). Initial values of velocities was generated randomly form the range $[-v_{\max}\ v_{\max}]$, where $v_{\max} = 0.12d_5$. If velocity value v_i generated according to (11) is higher than $v_{\max}$, it is set to $v_{\max}$, and if it is lower than $-v_{\max}$, it is set to $-v_{\max}$.

TS: $L = 210$, $a = 0.12d_5$.

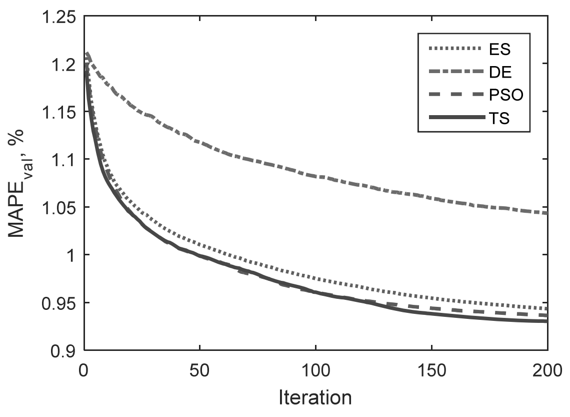

Fig.2. Convergence curves

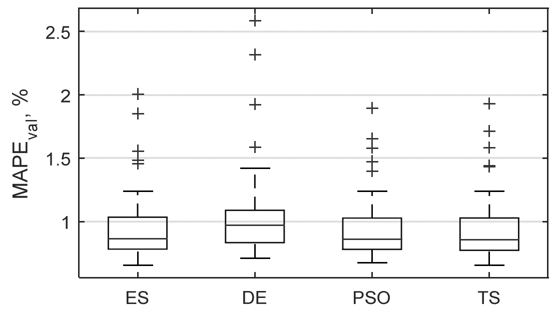

Fig.3. Error distribution

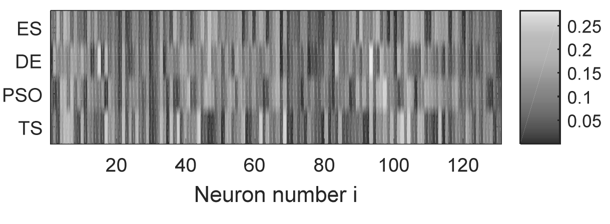

Fig.4. Solutions found by the algorithms for the last forecasting task (31 July 2004)

The population size in all cases was $M = 210$ and the total number of iterations was $I_{\max} = 200$. Components s_i of the initial solutions were generated randomly from the range

[0 1.2d_5]. The GRNN accuracy during training was evaluated in the local leave-one-out procedure, in which the validation samples are selected one by one from the set of 12 nearest neighbors of the current input test pattern. The mean error on validation samples ($MAPE_{val}$) is expected to be a good estimate of the error on test sample ($MAPE_{tst}$).

The convergence curves of the algorithms in Fig. 2 are presented. DE converges noticeably slower than other algorithms. The fastest convergence is observed for TS and PSO.

The error distribution (daily MAPE) is shown in Fig. 3. Note very similar distributions for TS, ES and PSO.

Solutions returned by the algorithms differ from each other. This is illustrated in Fig. 4, where the best solutions **s** found by the algorithms for one of the forecasting task is shown. For 61 forecasting tasks in 32 cases TS brought best results among all algorithms, ES in 15 cases, PSO in 14 cases, and DE did not bring a best result in any case.

Mean errors achieved on the validation samples and test samples in Table 1 are shown. As can be seen from this table, the best results for the validation samples do not translate into the best results for the test samples. This is because insufficient information about the target function hidden in the training points which are sparse distributed in n-dimensional space.

Table 1. Forecasting errors

Method	$MAPE_{val}$, %	$MAPE_{tst}$, %
ES	0.94	1.34
DE	1.04	1.08
PSO	0.93	1.18
TS	0.93	1.20

Conclusions

In this work GRNN model for load time series forecasting is learned using four stochastic optimization methods. Ability to escape from the traps of local minima to search for the global minimum is an important feature of these methods. However, they are usually very sensitive to their parameters and tuning the parameters is difficult, tedious and problem dependent. Evolution strategy, particle swarm optimization and tournament searching demonstrated similar effectiveness and showed the fastest convergence towards the solution.

It is worth noting, that tournament searching is the simplest one among examined stochastic optimization methods. It has an efficient and flexible algorithm which can be adapted to combinatorial, continuous and mixed optimization. In continuous version it has only two parameters controlling the global/local searching properties of the algorithm: the tournament size and the standard deviation used for generating trial solutions.

Author: *Grzegorz Dudek PhD, DSc, Czestochowa University of Technology, Institute of Computer Science, al. Armii Krajowej 17, 42-200 Czestochowa, Poland, E-mail: Dudek@el.pcz.czest.pl.*

REFERENCES

[1] Specht D.F., A General Regression Neural Network, *IEEE Transactions on Neural Networks*, 2 (1991), No. 6, 568-576
[2] Dudek G., Pattern Similarity-based Methods for Short-term Load Forecasting – Part 1: Principles, *Applied Soft Computing*, 37 (2015), 277-287
[3] Beyer H.G., Schwefel H.P., Evolution Strategies - A Comprehensive Introduction, *Natural Computing*, 1 (2002), No. 1, 3-52
[4] Storn R., Price K., Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces, *Journal of Global Optimization* 11 (1997), No. 4, 341-359
[5] Das S., Suganthan P.N., Differential Evolution: A Survey of the State-of-the-Art, *IEEE Transactions on Evolutionary Computation* 15 (2011), No. 1, 4-31
[6] Eberhart R. C., Shi Y. H., Particle Swarm Optimization: Developments, Applications and Resources, *Proc. IEEE Congr. Evol. Comput.* (2001), 81-86
[7] Eberhart R. C., Shi Y. H., Comparing Inertia Weights and Constriction Factors in Particle Swarm Optimization, *Proc. IEEE Congr. Evol. Comput.* (2000), 84-88
[8] Dudek G., Tournament Searching Method to Feature Selection Problem, *Proc. Artificial Intelligence and Soft Computing*, LNCS 6114, (2010), 437-444
[9] Dudek G., Tournament Searching Method for Optimization of the Forecasting Model Based on the Nadaraya-Watson Estimator, *Proc. Artificial Intelligence and Soft Computing*, LNCS 8468, (2014), 351-360