INFORMATION
SYSTEMS IN
MANAGEMENT

Information Systems in Management (2017) Vol. 6 (2) 119−130

EVOLUTIONARY COMPUTING IN OPERATIONAL RESEARCH FOR TWO-LAYER NEURAL NETWORKS

STANISŁAW PŁACZEK

Vistula University, Warsaw, Faculty of Computer Science

Considering the non-linear characteristics of the activation functions, the entire task is multidimensional and non-linear with a multimodal target function. Implementing evolutionary computing in the multimodal optimization tasks gives developers new and effective tools for seeking the global minimum. A developer has to find the optimal and simple transformation between the realization of a phenotype and a genotype. In the article, a two-layer neural network is analysed. In the first step, the population is created. In the main algorithm loop, a parent selection mechanism is used together with the fitness function. To evaluate the quality of evolutionary computing process different measured characteristics are used. The final results are depicted using charts and tables.

Keywords: evolutionary algorithm, neural network, optimization algorithm, mutation and crossover operator

1. Introduction

Genetic Algorithms (GA) and Evolutionary Algorithms (EA) are in some respect inspired by the process of natural selection. A given environment is filled with a population of individuals that strive for survival and reproduction. The fitness of these individuals represents their chances of survival. Between the concepts of natural evolution and computer calculation one can appoint the following relation [1]:

Environment – Solving problem
Individuals – Candidate solution
Fitness value – Quality or target.

Everybody should be very careful when interpreting the results achieved; taking into account the differences between a natural environment and a computer environment built with silicon and mathematics. From the historical point of view, three different implementations of the basic idea have been developed. In the USA, Fogel, Owen and Walsh introduced Evolutionary Programming (EP), while Holland called his method a Genetic Algorithm (GA); in Germany Rechenberg and Schwefel invented the Evolution Strategy. Since 1990, all streams following the general idea emerged as Genetic Programming (GP). Now the whole field of evolutionary computing is known as Evolutionary Algorithm (EA). In the article the last term is used.

In the article, an EA is implemented to a two-layer Artificial Neural Network (ANN) in the teaching process. An ANN has valuable characteristics, such as approximating any non-linear mapping and generalization, parallel and distributed computation, learning and adaptation. Both parallel and distributed computation especially correlate with the genetic and evolutionary algorithm structures. Evolutionary Algorithms are interpreted as a generalization of a genetic algorithm. In an EA, the evolution principle and inheritance are implemented, as well as using the appropriate data structure according to the solving task. For an ANN a real figure matrix is applied. Using this, the appropriated variations of operators are used.

2. Standard learning algorithm structure

The simplest ANN structure is described as a two-layer ANN (Fig. 1). The input vector data X are put into input neurons, which are multiplied by appropriate matrix $W1$ weight coefficients of an ANN structure. Using different types of activation function, the output signal of the first layer is calculated as vector U. Again this vector is multiplied by matrix $W2$ weight coefficients and after the activation function, the output vector Y is achieved. In the next step, this vector is compared with the teaching vector Z and the target function Φ value is calculated. As a teaching function, the minimum of the mean square error (MSE) is usually used. According to the ANN scheme structure, a set of formulas is calculated.
Stepping back from the left to the right, the target function is calculated:

$$\Phi = (Y - Z)^T * (Y - Z) \tag{1}$$

where:
Y – the output vector of a two-layer ANN,
Z – the teaching vector, one from the epoch set.

The ANN output *Y* has the following vector structure form

$$Y = F(W2 * U) \tag{2}$$

where:
F – the vector's structure of the activation function,
*W*2 – the output layer's matrix of the weight coefficients,
U – the internal vector that is the output vector of the hidden layer.

The vector *U* represents the output of the hidden layer. The hidden layer plays a fundamental role in the ANN learning algorithm. The input vector *X* is divided into vector *U* of the higher dimensionality in a new space.

$$U = F(W1 * X) \tag{3}$$

where:
F – the vector's structure of the activation function. Usually it is a sigmoid or a tanh function,
*W*1 – the hidden matrix of the weight coefficients,
X – the input vector, one from the epoch.

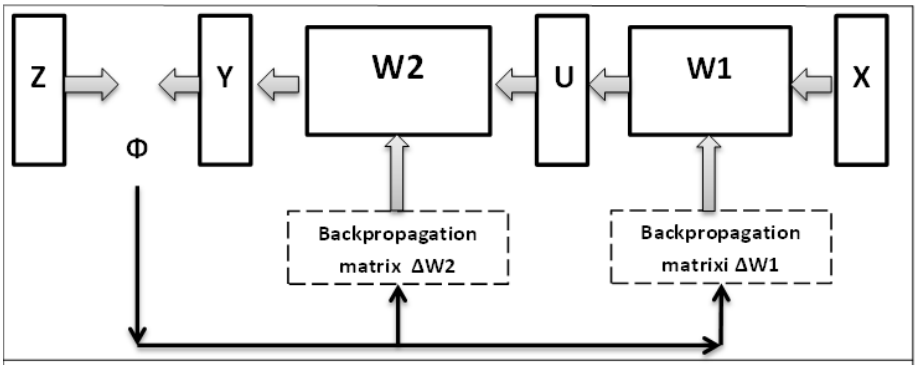

Figure 1. Backpropagation algorithm structure for a two-layer ANN

In the next step two differences are calculated, for *W*1 and *W*2 appropriately, and all weight coefficients are modified. So, the iteration is finished and the new input vector *X* and the learning vector Z from the epoch are used and the entire procedure is repeated. Only one model of matrix weight coefficients (*W*1, *W*2) is used (using the EA terminology: only one individual is used). In the EA, a set of individuals is generated. Evaluating the output vector *Y* is fulfilled in a parallel way, so the gradient calculation is unnecessary.

3. Evolutionary Algorithm structure

The basic model is different in an EA calculation. We have to define the set of ANN individuals with one input *Y* and the output learning vector *Z* (Fig. 2). These individuals are a unit of selection. Their reproductive success depends on how well they are evaluated by the target function. The more successful individuals have a higher ability to reproduce in the next generation. Additionally, mutations give rise to new individuals to be tested. Thus, as the iteration passes, there is a chance in the constitutions of the population. The whole set of individuals is known as a population. In an ANN every individual is represented by the weight coefficients, both matrices *W*1, *W*2, which connect the input signal *X* with the output *Y*. In the EA terminology, a matrix is defined as two chromosomes – an individual calculation entity. A chromosome contains a set of weight coefficients known as genes. Genes are expressed in natural figures. This is the basic difference between EA and GA. Remember, that in the GA all genes have bit representation.

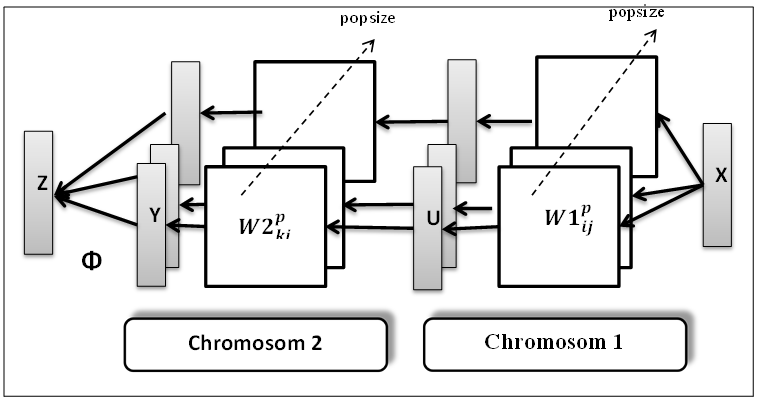

Figure 2. An EA algorithm structure for two chromosomes and a *popsize* population

Target function is a set of a *popsize* individual's target function.

$$\Phi = [\Phi^1, \Phi^2, \Phi^3, \ldots \Phi^{popsize}] \tag{4}$$

Every individual's target function is calculated according to formulas (5) and (2), (3)

$$\min \ \Phi^p = (Y^p - Z)^T * (Y^p - Z) \tag{5}$$

where:
$p = p^1, p^2, p^3, \ldots p^{popsize}$ – the set of individuals.

Thus, a set of genes creates a chromosome. According to Fig. 2, an EA structure contains the "number of individuals" dimensionality, *popsize*. The input vector

$X^p[0:N_0]$, in a parallel way is sending into the *popsize* matrix $W1$ → vector U → matrix $W2$ → to output vector Y. All output vectors $Y^p[0:N_2]$ are evaluated and the fitness function Φ^p for every individual is calculated. One of the most important and necessary parameters is the number of individuals (or parents) μ, and the number of offspring (children) λ. The decision is not simple. If one decides to increase the number of both the individuals and the offspring, the algorithm will look through the solving space better. But, on the other hand, this process is time-consuming and finding the final solution could last too long. Comparing Figures 1 and 2 one can see the main differences between a standard backpropagation algorithm and an EA structure. A backpropagation algorithm is serial. Solving space is indwelled by an algorithm using information contained in the target function gradient. An EA is a parallel algorithm. In a parallel way, the set of individuals indwells solving space, looking for the best individuals fitted to the target function (the solving task).

4. Evolutionary Algorithm's components

In literature, one can find many different variants of an EA. But there are common technics behind all of them. There is a population of individuals within some environment (giving a target function). These individuals compete using the natural selection (survival of the fittest). This, in turn, causes a rise of the population. Giving the target function (the quality function to be maximized), an algorithm randomly creates a set of candidate solution. The target function is applied as an abstract fitness measure. On the basis of these fitness values some of the better candidates are chosen to seed the next generation, which is fulfilled by applying recombination and mutation to them. Therefore, by executing the variation operators on the parents, a new set of candidates is generated (the offspring). A new target function value is evaluated and competition is created. For competition, a different algorithm is used – the fittest, the maximum age. This process can be iterated until a candidate with the maximum quality is found. To summarize, one can conclude [1]:

- EAs are population-based. They process a whole collection of candidate solution in a parallel way.
- EAs can use recombination, mixing information from two or more candidates and mutation for one candidate.
- EAs are stochastic using a lot of randomly generated data.

Below, the most important components of EAs are described:

- Individuals' representation. For an ANN a set of matrices will be applied.
- Evaluation function, target function or fitness function.
- Population dimensionality, *popsize*.

- Individual (parent) selection.
- Variation operators. Recombination and mutation to the matrix structure will be applied.
- Replacement (survival selection mechanism).

The general block scheme of the EA is given in Figure 3. It contains all the main components and the relations between them. Every block in the scheme could be realized in many ways and could be implemented by various algorithms, which are also described in Figure 3.

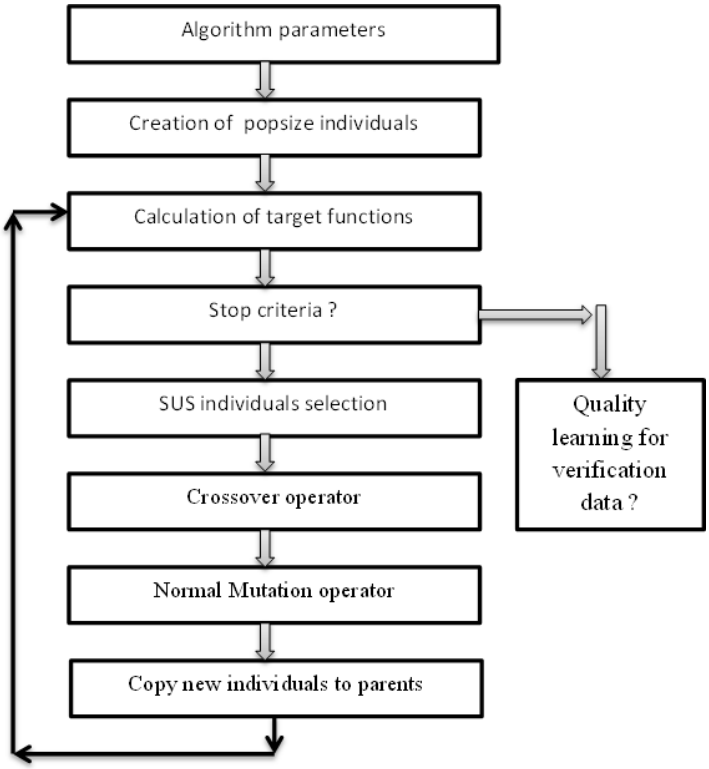

Figure 3. Scheme of Evolutionary Algorithm structure

4.1. The representation of an ANN weight coefficients

As stated above, an ANN layer structure is described as the weight coefficients matrices $W1$, $W2$. For the long input vector X, the hidden vector U and the output vector Y, the matrices' dimensionality could be big and contain a lot of weight coefficients.

For the input vector $N_0 = 50$, the hidden vector $N_1 = 80$ and the output vector N_2= 30, matrices' dimensionalities are:

$W1$= ||4080|| figures.

$W2$= ||2480|| figures.

Total number of genes = 6560. For the binary representation, the total bit number will be huge. The final conclusion is simple – using binary code for matrix weight coefficients representation is not adequate. In contrast, taking the real value matrix weight representation is the appropriate decision. From the calculation point of view, the coding and decoding step is omitted in each iteration. The algorithm works faster. The main role of the evaluation function, target function or fitness function is to represent the requirements the population should adopt to meet. It is the base for the selection step and it boosts improvements.

4.2. Evaluation function

As it is standard in an ANN teaching process, the minimum of the mean square error (MSE) is used (1).

It should be as minimal as possible. However, in an EA the maximum of the target function is required to use the proportional selection. The simplest method is to change the minimum to the maximum,

$$\max\{-\Phi(x)] = \min[\Phi(x)]. \quad (6)$$

But this simple method cannot guarantee that all $-\Phi(x) \geq 0$, which is required by the proportional selection. A better way is to add the constant number C to all fitness values, achieving

$$C - \Phi(x) \geq 0. \quad (7)$$

The problem is that it is hard to select the proper C value. If C is small, it cannot guarantee that $C - \Phi(x) \geq 0$ for all individuals. For an ANN teaching target function, the maximum of fitness value in the current population is calculated, which guarantees that:

$$\Phi_{\max}(\mathrm{x}) - \Phi(x) \geq 0. \quad (8)$$

In this way, the minimum optimization problem could be changed to:

$$\max\{\Phi_{\max}(\mathrm{x}) - \Phi(x)\}. \quad (9)$$

4.3. Population dimensionality

The role of a population is to represent a possible solution. In an ANN, the population is a multi-set of matrices and forms the unit of evolution. In almost all EA applications the population size is constant and does not change during the iteration process. The selection operators (two-step, parent selection and survival

selection) work on the population level. But, in contrast, the variation operators (crossover and mutation) act on parent individuals. A population should have an adequate property. The main parameter is known as diversity and measures the number of different solutions (12). Other statistical measures, such as variation (14) and entropy, are used. If the *popsize* population is big, diversity is usually better, but an algorithm needs more time to evaluate the target functions. A set of formulas is calculated to measure the EA's characteristics:

$$\Phi_{max} = \bigvee_{i \in popsize} \bigvee_{j \in popsize} \{ \Phi_{max} = \Phi_i \geq \Phi_j \} \tag{10}$$

$$\Phi_{min} = \bigvee_{i \in popsize} \bigvee_{j \in popsize} \{ \Phi_{max} = \Phi_i \leq \Phi_j \} \tag{11}$$

$$\Delta = \Phi_{max} - \Phi_{min} \tag{12}$$

$$\Phi_{aver} = \frac{1}{popsize} * \sum_{i=1}^{popsize} \Phi_i \tag{13}$$

$$\text{Var} = \frac{1}{popsize} * \sum_{i=1}^{popsize} (\Phi_i - \Phi_{aver})^2 \tag{14}$$

where:
Φ_{aver} – the average value of the fitness function for all population,
Var – the variation value as a diversity measure.

4.4. Parent selection mechanism

Selection process imitates natural selection by granting fitter individuals higher opportunity to be selected into the crossover process. Individual "i" in the current population has a fitness value Φ_i. Fitter individuals have more chances to be selected and a relative fitness value of individuals is calculated:

$$p_i = \frac{\Phi_i}{\sum_{i=1}^{i=popsize} \Phi_i} \tag{15}$$

In the program selection, an algorithm is simulated by the roulette and this process is known as roulette wheel selection (RWS). In this way, some individuals in the population will be selected more than once and some will never be selected. But, from time to time, the fitter individuals could not be selected by RWS. This process is known as *selection bias*. To avoid this for an ANN, *stochastic universal sampling* (SUS) is used, as suggested by Baker [2]. RWS has one arrow. If the arrow stops at area "i", an individual "i" is selected. RWS is carried out in a serial way. SUS has *popsize* uniformly distributed arrows (Fig. 4). If any arrow "i" stops at area "j", "j" individuals are selected. SUS works in a parallel way and could perform proportional selection without bias.

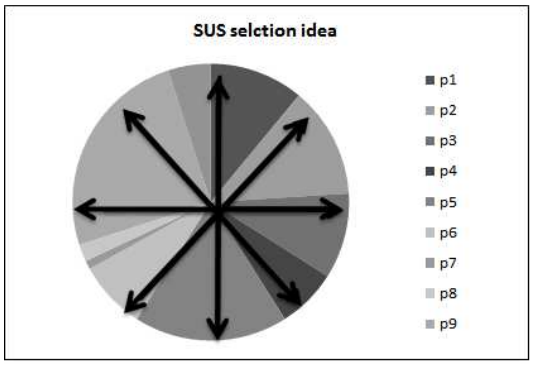

Figure 4. Stochastic Universal Sampling algorithm. *Source*: [5]

4.5. Mutation operator

In an EA implementation, only the mutation operator is proposed. Using the mutation probability parameter p_m, selected from the whole population, a gene could be modified by Uniform Mutation.

$$w1_{ij}^{p} = rand\,(a, b) \tag{16}$$

$$w2_{ij}^{p} = rand(a,\ b) \tag{17}$$

Where *a*, *b* are the upper and lower domain value, respectively. In this algorithm, one can define the domain parameters in an arbitrary way. A better result is achieved using Normal Mutation. For a normal distributed randomly, the possible of the mutant value will be in the range:

$$(w_{ij}^{p} - 3 \cdot \sigma\ ;\ w_{ij}^{p} + 3 \cdot \sigma) \tag{18}$$

Where σ is the standard deviation. Mutation is calculated by formula (19).

$$w1_{ij}^{p}(n+1) = \ w_{ij}^{p}(n) + \sigma \cdot N(0,1) \tag{19}$$

$$w2_{ij}^{p}(n+1) = \ w2_{ij}^{p}(n) + \sigma \cdot N(0,1) \tag{20}$$

5. Numerical example

A simple classification problem was investigated as an example for a two-layer ANN with configuration 3-7-1 (3 input neurons, 7 hidden, 1 output). Data located in a three-dimensional space have to be classified in two categories: −1 and 0.

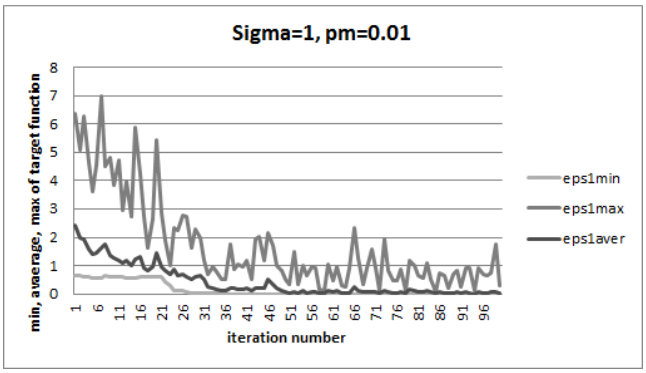

Figure 5. Three population characteristics: minimum, average and maximum target function values as a function of iteration number. Mutation parameter: sigma = 1

An Evolutionary Algorithm used selection and a mutation operator as main algorithmic forces. A cross operator was ignored. For a mutation operator, two parameters were used: sigma, according to formulas (19) and (20), modified the weight matrices $W1$, $W2$ coefficients with probability $p_m = 0.01$. To study the learning process' dynamic characteristics, only sigma was modified. In Figure 5, the entire learning process can be divided into two parts. At the beginning, all three parameters (minimum, average, maximum) are greater than zero. After about 40 iterations, a set of population achieves the minimum target function value and the learning process could be finished, but the average value is still greater than zero. After 80 iterations, the algorithm was stopped.

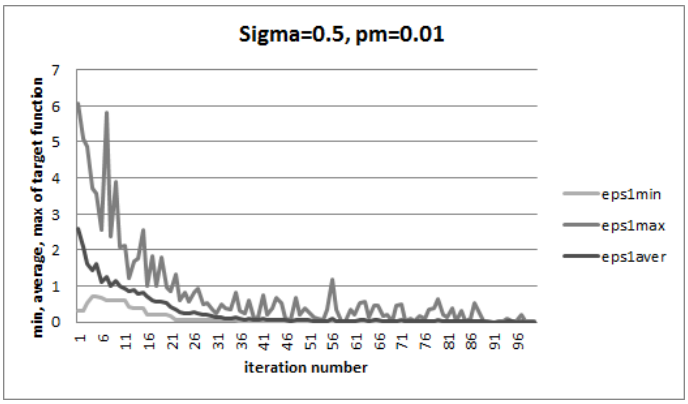

Figure 6. Three population characteristics: minimum, average and maximum target function value as a function of the iteration number

In Figure 6, for sigma = 0.5 all characteristics of the learning process go more smoothly. The average value achieves zero in an asymptotic way. In Figure 7, a variation operator is shown.

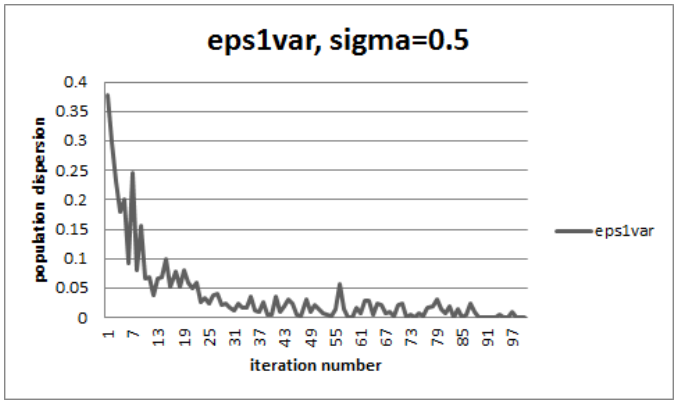

Figure 7. Variation as a measure of selection pressure.

The variation operator characterizes the population's diversity and, for sigma = 0.5, a population very quickly concentrates around the target function (the minimum value is close to zero). For sigma = 0.5 we can observe that the selection pressure is too high. As far as the algorithm's stability is concerned, it is not a good strategy as it converges too fast.

6. Conclusion

An ANN calculates the output value Y using all the neurons in layers (hidden and output), which are working in a parallel way. The modern computers with multiprocessors and programming languages have the tools to use ANNs in a wide area. An EA, using real figures value in genes, chromosome and population representation as the basic for its structure, can use the same computers and programming features. So, the connection of these two tools increases the final calculation power and speed. The article, in a very abbreviated form, describes all the main features and elements of an EA structure. In the calculation example, SUS selection process and Normal Mutation algorithms were used. As the replacement mechanism the simplest algorithm was used, too. The *popsize* individuals (parents) generate the same offspring number, so the population is constant during the iteration process $\mu = \lambda$. Using a threshold activation function, an algorithm needs less iteration to achieve the final target function value. Using a standard sigmoid function, a backpropagation algorithm needs more iteration to achieve the saturation value.

There are two important issues in the evolution process: population diversity and selective pressure. These factors are related: an increase in the selective pressure decreases the diversity of the population, and *vice versa*. Strong selective pressure supports the premature convergence. Future work will be concentrated on the adaptation algorithm for Normal Mutation. The sigma parameter has to change its value. When the learning algorithm starts, the sigma should be quite big and should decrease during iteration.

REFERENCES

[1] Eiben A.E., Smith J.E.: *Introduction to Evolutionary Computing*, Second Edition, Springer 2003, 2015.

[2] Michalewicz Z.: *Genetic Algorithm + Data Structure = Evolutionary Programs*, Springer-Verlag Berlin Haidelberg 1996.

[3] Montana DJ, Davis L.: Training Feedforward Neural Network Using Genetic Algorithms. Proceedings of the 1989 International Join Conference on Artificial Intelligence", Morgan Kaufmann Publishers, San Mateo CA 1989.

[4] Goldberg David E.: *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Veslay Publishing Company, Inc. 1989.

[5] Xinjie Yu, Mitsuo Gen: *Introduction to Evolutionary Algorithm*, Springer London 2010.

[6] Nolfi Stefano, Floreano Dario: *Evolutionary Robotics*, The MIT Press, Cambridge, Massachusetts, London 2000.