Comparison between Different Methods for Developing Neural Network Topology Applied to a Complex Polymerization Process

Silvia Curteanu, Florin Leon, Renata Furtuna, Elena Niculina Dragoi, Neculai Curteanu

In this paper, three methods for developing the optimal topology of feed-forward artificial neural networks are described and applied for modeling a complex polymerization process. In the free radical polymerization of styrene, accompanied by gel and glass effects, the monomer conversion and molecular masses are modeled depending on reaction conditions. The first proposed methodology is an algorithm which systemizes a series of criteria and heuristics on neural networks modeling. The method is laborious, but the practical considerations, structured in a 6-steps algorithm, and the criterion and formula used for calculating the performance indices give the method a real chance of obtaining a neural network of minimum size and maximum performance. The next two methods belong to evolutionary techniques and they are based on a classical genetic algorithm and differential evolution algorithm. They automatically develop the neural network topology by determining the optimal values for the number of hidden layers, the number of neurons in these layers, the weights between layers, the biases of the neurons and the activation functions. For all the three methods, a combination between training and testing errors was considered in order to evaluate the performance of the developed neural networks and to choose the best one among them. The relative percentage errors calculated in the validation phase registered good values, under 7%. A comparison between these methods pointed out both advantages and disadvantages, but, even if they lead to different network architectures, accurate results were obtained and, consequently, near optimal neural network topologies were developed.

I. INTRODUCTION

Among the known types of neural networks (NN), the feed-forward neural networks are the mostly used because of their simplicity, flexible structure, good qualities of representation, and their capability of universal approximation.

In the open literature, various approaches provide practical considerations for developing neural networks, or implement different methods which lead to the optimal topology of the neural networks. For instance, Fernandes and Lona [1] present a short tutorial helping to the neural network selection and training, along with some applications in the field of complex polymerization reactions.

The methods used to develop NN optimal architecture may be classified into the following types: 1) the trial and error method, which consists in successive tests relying on the development of several configurations of the neural networks and the evaluation of their performance [2]-[4]; 2) empirical or statistical methods, which study the influence of different internal parameters of NNs, choosing their optimal values depending on the network performance [5]-[10]; 3) hybrid methods, such as the fuzzy inference, in which a network can be interpreted as an adaptive fuzzy system or it can operate on fuzzy instead of real numbers [11]; 4) constructive methods and/or pruning algorithms, which add and/or remove neurons or weights to/from an initial architecture, using a pre-specified criterion to show the manner that these changes should affect the network performance [12]-[18]; 5) evolutive strategies, which are based on the search within the space of NN topologies, by varying the NN parameters on the basis of genetic operators [19]-[24].

A comparison between the methods 4) and 5) was made in a paper of Islam and Murase [25] where it was pointed out that the main difficulty of evolutionary algorithms is that they are quite demanding in both time and user-defined parameters. In contrast, non-evolutionary algorithms require much smaller amounts of time and user-defined parameters. Other opinions [22], [23] are favorable to the evolutionary methods based on the main consideration that they have good chances to achieve the optimal configuration.

In the present paper, three methods for determining the neural network topology are applied and compared using a complex nonlinear process as case study. These methods are: an optimization methodology based on a systematized trial and error applied for NN parameters (OMP) and two evolutive strategies based on differential evolution (DE) and a genetic algorithm (GA).
OMP [10] represents an iterative algorithm which synthesizes a series of practical considerations on neural network modeling, GA is based on the representation of NN parameters into chromosomes and DE is also based on principles of evolution strategy. In all cases, both the architecture and the optimal internal parameters of a neural network are determined.

A comparison between the three methods is made from different points of view: the accuracy of the results, the accessibility of the method, the modelling goal, and the dependence on the approached case study.

The novel elements of this paper are mainly related to the application of neural network based modeling to a complex chemical process. It was emphasized that neural networks constitute a better alternative compared to classical mathematical modeling of a polymerization process and, also, that the method for determining the network topology depends on the initial set of data (type of application). The neural modeling usefulness and the accuracy of its results are dependent on the development of a rigorous modeling methodology, in which the main element is determining the optimal network topology.

I. DEVELOPING NEURAL NETWORK TOPOLOGY

A. Data Base

The free radical polymerization of styrene performed by suspension technique is the case study of this approach. A complete mathematical model based on the kinetic diagram was elaborated and solved in one of our previous work [26].

Modeling with neural networks is an acceptable alternative to the phenomenological model due to a series of difficulties affecting the latter. Writing the conservation equations for the elements in the reaction mixture leads to an infinite system of differential and algebraic equations because the molecules include different number of structural units. A special solving procedure is applied using the distribution moments of the concentrations. In addition, the diffusion phenomena (gel and glass effects) are one of the most difficult parts in the modeling action. The equations for quantifying these effects contain many empirical parameters which have to be determined by a fitting procedure. Consequently, the advantages of a mechanistic model which is based on the phenomenology of the process are diminished and the empirical models like neural networks become recommended alternatives. That is why the present paper developed and applied neural network modeling methodologies.

The mathematical model [26] was the simulator for producing the working data base for neural network modeling used to predict the changes of monomer conversion and molecular weights depending on initiator concentration, temperature, and reaction time.

Data quality and quantity are essential for modeling with neural networks. In this case, the collected data was chosen to cover the whole field of interest for the studied process and to be uniformly distributed in it. Thus, for the initiator concentration and temperature, the ranges specific to the suspension polymerization of styrene were 10 – 55 mol/m³ (variation step 5) and 60 – 90 °C (variation step 10), respectively. Regarding the reaction time, the interval was 0 to 2000 minutes, because it was taken into account the fact that for low concentrations of initiator and low temperature, the reaction time is longer. For example, for a concentration of initiator of 15 mol/m³ and a temperature of 60 °C, the maximum conversion is achieved in 2000 minutes, and for 55 mol/m³ and 90 °C, the reaction time was 600 minutes. In addition, the reaction step of 10 minutes guarantees the correct reproduction of the variation curves for conversion and molecular weight, provided that a self-acceleration of the reaction appears at some point, dependent on initial working conditions. Therefore, sudden jumps occur on the curves of conversion and molecular weights.

Consequently, three variables are the inputs of the neural models: initiator concentration, \( I_0 \), temperature, \( T \), and reaction time, \( t \), and other three variables are the outputs of the networks: monomer conversion, \( x \), numerical average molecular weight, \( M_n \), and gravimetrical average molecular weight, \( M_w \).

B. Methodology for determining the NN topology based on optimizing neural network parameters (OMP)

Whereas the number of neurons in a hidden layer was a parameter chosen for optimization by all researchers approaching this subject, the other selected neural network parameters varied from a researcher to another. Vlahogianni et al. [8], Kim and Yum [27] and Packianather [28] took into account the step size and the momentum term, and also the interactions between these parameters. Sukhomylia and Tannock [6], Kim and Yum [27], Wang et al. [29] have observed that the activation functions, the size of the training set, and the size of the testing set have a significant effect on neural network performance. The number of hidden layers, the learning rules, the error function, and the noise factors were other optimized neural network parameters [27], [29].

In this paper, the architecture of the neural network chosen for optimization was the multilayer perceptron (MLP) trained with the back-propagation algorithm because it is the most widely used and effective in solving classification and prediction problems.

The most important practical considerations on which the optimization methodology is based are the following:

- the optimum number of hidden layers has been found to be less than the number of inputs [30];
- the activation function must be chosen based on the kind of data used and the type of layer: the input layer has no activation function, the hidden layers must have nonlinear functions (hyperbolic tangent function, \( tanh \), is preferred), and the output layer is recommended to have activation functions adapted to the distribution of the output data;
- the optimum number of hidden neurons have to satisfy the following equations:

\[
N_w \leq \frac{1}{10} \cdot N_t \cdot O, \quad (1)
\]

or, if the training data set is too small:

\[
N_w \leq \frac{1}{2} \cdot N_t \cdot O \quad (2)
\]
and for a two hidden layers neural network:
\[ H_1 \geq 3 \cdot H_2, \]  
where \( N_w \) is the number of weights, \( N_i \) is the number of training vectors, \( O \) is the number of outputs, \( H_i \) is the number of neurons in the first hidden layer, and \( H_2 \) is the number of neurons in the second hidden layer;

- the optimum number of training epochs can be determined through cross-validation, the method also used for stopping network training;
- the learning rate for the neurons in the output layer must be smaller than the one for the hidden layers;
- a positive momentum value leads to faster learning, with a small learning rate; the momentum term must be less than 1 for convergence.

The steps describing the algorithm of finding the optimum parameters for a feed-forward neural network are:

1. Finding the optimum number of neurons in the hidden layer for one hidden layer neural network.
2. Finding the optimum value for the learning rate.
3. Finding the optimum value for the momentum term.
4. Finding the optimum activation function for the output layer.

5. Finding the optimum number of neurons in the hidden layers for a two hidden layers neural network.

6. Optimizing the parameters for the two hidden layers neural network following the steps 2 – 4.

For the quantification of the network performance the following formula was used:
\[ perf\_index = r - (MinMSECV + NMSEtesting). \]  
where \( r \) is the linear correlation coefficient at testing, \( MinMSECV \) is the minimum mean squared error at cross-validation, and \( NMSEtesting \) is the normalized mean squared error at testing. The greater the value of \( perf\_index \) is, the better the performance of the network.

Data from the input file for the neural network was randomized and divided into 75% for the training and cross-validation and 25% for the testing phase. The data for training and cross-validation was again divided into 80% for training and 20% for cross-validation. Thereby, for the particular problem studied, for a data set of 3494 exemplars, 2097 exemplars were used for training, 524 for cross-validation, and 873 for testing.

C. Determining the NN topology using the GA based method

Another method applied in this paper for developing the optimal topology of a neural network uses a genetic algorithm based on three reasons: 1) The GA operates on a coded form of the parameters, that means a real chance to obtain the global optimum, independent of the continuity of the function or the existence of its derivative. 2) The procedure uses a set of start points, not a single point of the solution space, thus increasing the chances of avoiding the local minima. 3) The genetic operators are stochastic and not deterministic in nature [23].

Castillo et al. [31] pointed out different ways to apply GA to design NNs: search over the topology space, where a population is initialized with different hidden layer sizes, search for the optimal set of weights of a pre-established topology net, where GA encodes only the parameters of the hidden layer, search for the optimal learning parameters, including weights, but having pre-established the number of neurons and the connectivity between them, genetic approaches that modify the back-propagation algorithm. Recently, Dam and Saraf [22] developed a methodology for designing the network architecture using a GA, taking into account the number of hidden layers, the number of nodes in each layer, the connectivity, and the activation functions. Gerken et al. [21] presented a novel approach for the neuron model specification using GA to develop simple firing neuron models consisting of a single compartment with one inward and one outward current. Benardos and Vosniakos [23] assume that the most critical elements of a feed-forward NN architecture are the number of hidden layers and hidden neurons which affect the generalization ability of the NN model. Castillo et al. [31] present a method based on GA and BP (back-propagation) that searches over the initial weights and number of neurons in the hidden layer of a MLP with a single intermediate layer.

The chromosome structure in our GA method applied for determining the NN topology contains genes that encode the following information: the number of hidden layers (1 or 2), the number of neurons in the hidden layers, the weights between the NN layers, and the biases of the neurons for the hidden and output layers. All this information is coded as real numbers in the chromosomes.

This representation has both advantages (the simplicity of the approach, as the GA also accomplishes the finding of the optimum topology and the training of NN), but also some disadvantages (a long training time because of the greatly increased size of the search space needed to determine the optimal NN topology).

The GA fitness function was the mean square error (MSE) at training and testing.

The selection technique in GA was the method of ranks, in which the selection probability of the parents is proportional to their rank within the population, based on their fitness.

As the representation of the solutions is on real numbers, we used the arithmetical crossover with the probability of 0.8. This technique produces linear combinations of the parent values. In order to encourage (and accelerate) the search process, a relatively big value of the mutation rate was employed. The mutation resets the selected gene with a uniform value from the definition interval, for weights and biases taking values from -500 to 500.

D. Determining the NN topology using the DE based method

Differential Evolution (DE) is an optimization method based on principles of evolution strategy. The main idea of this technique is to provide new items to a pool of potential solutions by mathematically manipulating the existing ones. Extensive work was done to train the neural networks using the DE algorithm, but finding the best architecture by using it is a mainly unexplored area. This method was chosen because it is an algorithm for global optimization over
continuous spaces and it can be applied to global searches within the space of a typical feed-forward neural network [32].

Plagianakos et al. [33] used the DE algorithm to train neural networks with discrete activation functions. Subudhi and Jena [34], [35] used a hybrid method to train a neural network for nonlinear system identification, combining differential evolution and Levenberg Marquardt algorithms. An attempt to determine a partial architecture of a NN using DE was made by Lahiri and Khalfe [36]. Using the DE algorithm, they constructed a network by determining the optimal number of neurons in the hidden layer, the weights, and the activation function. It was a partial determination because they constrained the architecture at only one hidden layer.

There are many versions of DE algorithm and, in this paper, we chose to work with the DE/rand/1/bin version where rand indicates how the target vector is chosen, l represents the number of vectors that contribute to the differential, and bin is the type of distribution.

In our paper, DE starts with a population of vectors representing the potential solution:

\[ X = \{x_1, x_2, ..., x_n \} \]

where \( X \) is a vector containing all the potential solutions and \( x_1, x_2, ..., x_n \) are vectors that contain the coded neural network for the studied problem.

The initial vector is generally initialized by randomly generating a set of vector solutions. After initialization, a set of intermediary solutions called mutants are created on the basis of the existing ones. Although there are different methods to create the mutants, the technique in the chosen variant of DE is to add a scaled, randomly sampled vector difference to a third one:

\[ m_{i,j} = x_0,j + F \cdot (x_{1,j} - x_{2,j}), j = 0,k \]

where \( m_i \) represents the mutant vector, \( x_0 \) is the base vector, \( x_1 \) and \( x_2 \) are the difference vectors, \( F \) is the scale factor. The scale factor controls the rate at which population evolves. The base vector is different from the target vector, here the target vector being indicated by the \( j \) index of the mutant. The \( x_{0,j} \), \( x_{1,j} \) and \( x_{2,j} \) vectors are randomly selected once for every mutant. The \( k \) index represents the total number of the \( m_i \), \( x_{0,j} \), \( x_{1,j} \), and \( x_{2,j} \) vectors and the \( j \) index passes through each element starting from the position 0.

After the creation of mutant vectors, the current population is recomposed with the mutants to create a trial population:

\[ u_{i,j} = \begin{cases} m_{i,j} & \text{if } \text{rand}(0,1) < \text{cross_rate} \\ x_{i,j} & \text{otherwise} \end{cases} \]

where cross_rate represents the crossover probability and controls the fraction of parameter values that are copied from the mutant.

The next step is the selection. If the trial vector \( u \) has a better fitness function than the target vector, it will replace it in the next generation pool of solutions. Using this principle, only the best solutions are chosen.

After that, the current generation is replaced by the next generation and the mutation, crossover, and selection steps described above are repeated until a stop criterion is reached. In our application, the stop criterion is the mean square error reaching a very low level or the number of epochs reaching a preset value, which ever comes first.

An important aspect in modeling the architecture and training of a neural network using DE is coding the neural network. A floating point vector was created that contains all the information from the neural network, as can be seen in Fig.1. In this figure, \( NL \) represents the number of hidden layers, the only acceptable values being \( \{0, 1, \text{ and } 2 \} \). \( H_j \) represents the number of neurons in the first hidden layer, \( H_2 \) the number of neurons in the second hidden layer. The parameters \( w_1, ..., w_n \) indicate the weights for each neuron in the neural network, \( b_1, ..., b_n \) the biases and \( a_1, ..., a_n \) the activation functions.

![Fig. 1. The content of a solution vector in DE method](image)

In the current application, the fitness function is a combination of \( \text{MSE}^{\text{training}} \) and \( \text{MSE}^{\text{testing}} \), and the neural network parameters were optimized so that the maximization of the fitness functions is obtained.

### II. RESULTS AND DISCUSSION

By applying the OMP method, a series of simulations were performed using the trial and error method, through the "Vary A Parameter" training process [10], following the algorithm described in section II.B and starting with a single hidden layer neural network. This provided the best network with optimized parameters. Every network was tested using the test data set. Finally, the network with the best performance at training and testing was chosen for further optimization.

The heuristic rule (1), applied for a single hidden layer neural network, limited the number of hidden neurons at 105 (\( H_1 \leq 105 \)). Consequently, the number of neurons in the hidden layer was varied in the interval 5 – 105, with step 5. The maximum number of training epochs was set to 1000.

Some of the performance indices obtained through the simulations for optimizing the number of hidden neurons of a single layer neural network are presented in Table I. The best performance was reached for a number of 45 hidden neurons (the last line in Table I). The best number of training epochs found after the cross-validation process was 1000.

The learning rates and the momentum terms were varied in the interval 0.1 – 1. The neural network performance obtained after the simulations (average perf_index = 0.8525) did not increase, so the learning rates and the momentum terms were kept to the initial values (1 – the learning rate for hidden layer, 0.1 – the learning rate for the output layer, 0.7 – the momentum term for hidden and output layers).

The values of the input data were scaled for the activation function of the hidden layer (the hyperbolic tangent) with values from -0.9 to 0.9. The output values were scaled according to the activation function used for the output layer.
In order for the data used to train and test the neural network to be at the same scale, the values obtained with the simulator for molecular weights were divided by 1000. This is necessary because it was observed that the neural networks cannot learn well if data used for training are not in the same range/scale.

Because the studied problem was a regression problem, and the desired response was a continuous function of the input, the recommended output activation function was the linear function. Nevertheless, the simulations with linear functions did not provide a better performance for the neural network, the maximum average perf_index obtained being 0.8513, whereas the average perf_index of the network before the optimization of the activation function for the output layer was 0.8534.

The optimization algorithm ended with the development of an optimized two hidden layers neural network and its comparison with the previous optimized one hidden layer neural networks.

For the optimization of the number of hidden neurons in the two hidden layers, the heuristic rules (1) and (3) indicated that \( H_2 \leq 12 \). Starting with this statement, taking into account the number of weights, and applying the "Vary A Parameter" training process, the best two hidden layers neural network proved to be a MLP with 21 neurons in the first hidden layer and 6 neurons in the second hidden layer, with an average perf_index of 0.8772 and 207 weights (the last line in Table II). Some results of the simulations can be observed in Table II.

Further optimizations on the parameters of the two hidden layers neural network revealed that a learning rate of 0.9 for the first hidden layer, one of 0.2 for the second hidden layer, and a learning rate of 0.07 for the output layer provided an average perf_index of 0.8934. Furthermore, it was observed that a momentum of 0.3 for the first hidden layer and one of 0.7 for the second hidden layer and the output layer, increased the neural network performance to an average perf_index of 0.9060.

The simulations for optimizing the activation function for the output layer did not lead to a better performance of the network. The greater average perf_index was obtained for the linear function, namely 0.8648.

By comparing the performance indices obtained for the single hidden layer neural network (average perf_index = 0.8534) and the two hidden layers neural network (average perf_index = 0.9060), respectively, the latter turned out to be the best.

The second method applied for developing the topology of a neural network used to model the styrene polymerization process was based on a GA methodology.

The results of GA are often influenced by the following parameters: the dimension of initial population (dim_pop), the number of generations (no_gen), the mutation rate (mut_rate), and the crossover rate (cross_rate). The method described in Section C is applied using different values for the GA parameters. Table III shows some neural network topologies obtained by changing these parameters, along with MSE in the training and testing phases. The column “topology” shows the number of neurons in one or two hidden layers.

The best result was considered a neural network 3:20:7:3 having the smallest error in the testing phase (0.113737, the line marked in bold in Table III). The optimum values found for GA parameters (for which the best neural network was achieved) were dim_pop = 100, no_gen = 100, mut_rate = 0.2, and cross_rate = 0.8.

A limitation of the number of hidden neurons was made within our GA: 30 neurons for the first hidden layer and 10 neurons for the second hidden layer were considered enough for the complexity of our problem. The bigger these values, the greater the search space of the GA, but also the greater the search time. The majority of applications from the field of polymerization reactions proved that the maximum number of hidden layers and neurons established in our procedure is enough for obtaining an accurate prediction.

In the case of the DE method, the following parameters are used: the crossover factor (cross_rate), the scaling factor (F), and the number of epochs. The efficiency of the algorithm is very sensitive to these parameters [37]. Hoenen et al. [32] consider that F and cross_rate affect the convergence speed and the robustness of the search space, and the optimal values of these parameters depend on the objective function characteristics and the population size parameter. Thus, the determination of the optimal parameter values is dependent on the problem being solved.

The role of the cross_rate parameters is to exploit the decomposability property of the problem, in case it exists, and to provide the diversity of the possible solutions. Price et al. [38] showed that all functions could be solved with cross_rate in the range 0 – 2 or 0.9 – 1. We tested cross_rate...
by variation of its values in the interval 0.9 – 1.

Like cross_rate, the F parameter takes values between 0 and 1, Price et al. [38] considering that F can take values bigger than 1, but it has been observed that no function tested required F > 1. Zaharie [39] demonstrated that a lower limit for F (called Fcritical) exists, in which case the population can converge even if the selection pressure does not exist. In practice F must be greater than Fcritical and, in our case, we tested F between 0.6 and 0.9.

The number of epochs represents the generations the algorithm will run. For the values of the epochs that are too low, the results are in local minima. If the values are too high, the computation time of the algorithm is too long but, for 5000 epochs our application had the results in approximately 2.5 hours.

As shown in the Table IV, the maximum fitness value was obtained when cross_rate = 0.99, F = 0.7, and the number of epochs = 5000.

<table>
<thead>
<tr>
<th>cross_rate</th>
<th>F</th>
<th>Epochs</th>
<th>Fitness</th>
<th>Topology</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.99</td>
<td>0.7</td>
<td>500</td>
<td>7.66</td>
<td>10</td>
</tr>
<tr>
<td>0.99</td>
<td>0.7</td>
<td>1000</td>
<td>13.55</td>
<td>6</td>
</tr>
<tr>
<td>0.99</td>
<td>0.7</td>
<td>2000</td>
<td>16.63</td>
<td>4</td>
</tr>
<tr>
<td>0.99</td>
<td>0.7</td>
<td>3000</td>
<td>20.79</td>
<td>4:2</td>
</tr>
<tr>
<td>0.99</td>
<td>0.7</td>
<td>4000</td>
<td>25.05</td>
<td>6</td>
</tr>
<tr>
<td>0.99</td>
<td>0.7</td>
<td>5000</td>
<td>27.08</td>
<td>9:7</td>
</tr>
</tbody>
</table>

Table IV: Results of the Simulations for F=0.7 and Cross_rate=0.99

A bigger network can better predict the desired parameters, but a coded bigger network has a bigger vector associated with it, which determines the increase of simulation time. A favorable compromise is reached by limiting the number of nodes in the first hidden layer at 20, and the nodes in the second hidden layer at 10.

Different results are obtained with the three methods, thus a comparative evaluation is necessary.

Table V presents relative errors obtained by applying the three methods, calculated with the following formula:

$$E_r = \frac{P_{desired} - P_{net}}{P_{desired}} \cdot 100,$$

where $P_{desired}$ represents $x$, $Mn$ or $Mw$ as targets of the neural network, and $P_{net}$ is the prediction of the network for each of the three output variables.

Besides errors, Table V also presents the correlations between the data used for testing (a number of 524 data) and the predictions of the three neural networks selected through the three methods (see the first column of the table).

Three types of activation functions were used: step, linear and bipolar sigmoid, each being coded using integer numbers. All the networks parameters excluding the number of inputs and outputs are internally changed by the algorithm described above. In the case of integer coded values of the activation function, the algorithm chooses the closest integer values of each of the $a_n$ parameters contained in the coded vector.

Fig. 2, Fig. 3, and Fig. 4 show the testing phase of the three methods based on the same data set. Only 10 data from the testing data set were chosen for the graphical representation because of clarity reasons. In these figures, the differences between the desired values and those provided by the neural networks become visible for the individual points of the testing set. Once again, it can be noticed that the three methods give comparable results whose accuracy can be considered very good.

Table V: Relative Errors and Correlations Obtained Through the Three Methods Applied for Developing Neural Network Topologies

<table>
<thead>
<tr>
<th>Average relative errors (%)</th>
<th>Correlations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$ $Mn$ $Mw$ $x$ $Mn$ $Mw$</td>
<td></td>
</tr>
<tr>
<td>OMP 3:21:6:3</td>
<td>0.044 5.480 7.018 0.999 0.989 0.988</td>
</tr>
<tr>
<td>GA 3:20:7:3</td>
<td>0.811 2.233 6.130 0.999 0.998 0.994</td>
</tr>
<tr>
<td>DE 3:9:7:3</td>
<td>0.787 2.437 6.385 0.999 0.993 0.979</td>
</tr>
</tbody>
</table>

Fig. 2: Some values for conversion: desired vs. obtained conversion in the testing phase through the neural networks designed with the three methods (OMP, GA, DE).

A comparison between these methods will take into account both the error values, but also other factors such as the accessibility of the method (execution time, algorithm complexity etc.) or the purpose to which the model is intended (prediction, monitoring, or control).

Fig. 5 shows the variation of monomer conversion and molecular masses with time, obtained with MLP(3:21:6:3) for training and validation data, and demonstrates that neural network modeling is an adequate procedure for rendering the process under study (the critical point where a sudden increase takes place is well modeled).

All the three methods have acceptable results in the testing phase, with percent relative errors less than 7%.

The three methods have both advantages and disadvantages. DE and GA provide better predictions for the molecular weights, $Mn$ and $Mw$, whereas the OMP method leads to very small errors for conversion in the testing phase.

In terms of accessibility, DE and GA methods, once implemented, are easy to use because the execution of the software program provides the optimal network topology, training and testing errors, and predictions over the training and testing data. However, the runs of the program should be repeated because of the stochastic nature of the algorithms. Furthermore, one cannot say precisely that the results of
these algorithms are optimal networks because there is always the possibility of reaching a local optimum. The parameters of these algorithms control the accuracy of the results, so optimal values have to be chosen for these parameters in order to obtain global optima.

The main advantages of DE and GA are the capability to work with noisy or nonlinear functions and the ability to find the global optimum, although the objective function can have multiple local optima. The biggest disadvantage is the fact that they can get stuck in local optima if the correct parameter values are not chosen.

The OMP method is more laborious, but the systemized practical considerations on modeling a neural network (structured in a 6-steps algorithm) and the criteria and formula used for calculating the performance indices give this method a real chance to obtain a neural network of minimum size and maximum performance.

In addition, the reason of testing three methods was related to the intention of developing a general methodology useful for different applications. Our previous experience showed that the method for obtaining the best neural network topology depends on the set of initial data (type of process). For example, DE method provided better results for classification problems; here, for styrene polymerization, OMP and GA worked better.

OMP method was developed in the neural network simulation environment NeuroSolutions, and it is very useful for beginners in neural modeling. It does not require any other knowledge about artificial intelligence tools and it provides the user with total control over the steps of the parameters optimization technique. Moreover, the performance indices formulated for the quantification of the obtained neural network performance outline an effective guideline at every step of the optimization methodology.

The two evolutionary methods used for developing the near optimal topology of neural networks (GA and DE) were implemented in C# programming language, and then the programs were integrated into a software product with a graphical user interface. This interface is simple to use, but the user must know the means to find the specific optimal parameters for each problem.

The software which implemented the modeling methodologies can be easily extended for different maximum number of intermediate layers and neurons and for other experimental data and applications.

III. CONCLUSION

The results of this paper will be improved and continued on the following directions: to quantify and classify all NN parameters which define and significantly influence the topology, to test other different objective functions, to improve the effectiveness of the algorithms in finding a NN optimal architecture, and to approach other complex case studies as tests for the developed methods.

REFERENCES


