13

# Pharmaceutical Drug Design Using Dynamic Connectionist Ensemble Networks

Ajith Abraham[1], Crina Grosan[2] and Stefan Tigan[3]

[1] Norwegian Center of Excellence, Center of Excellence for Quantifiable Quality of Service, Norwegian University of Science and Technology, O.S. Bragstads plass 2E, Trondheim, Norway
ajith.abraham@ieee.org

[2] Department of Computer Science, Babes-Bolyai University, Cluj-Napoca, 3400, Romania
cgrosan@cs.ubbcluj.ro

[3] University Iuliu Hatieganu, Faculty of Medicine, Department of Biostatistics and Medical Informatics, Cluj-Napoca, Romania
stigan@umfcluj.ro

**Summary.** This article presents a dynamic ensemble neural network model for a pharmaceutical drug design problem. Designing drugs is a current problem in the pharmaceutical research domain. By designing a drug, we mean to choose some variables of drug formulation (inputs), for obtaining optimal characteristics of drug (outputs). To solve such a problem, we propose a dynamic ensemble neural network model and the performance is compared with several neural network architectures and learning approaches. The idea is to build a dynamic ensemble neural network depicting the dependence between inputs and outputs for the drug design problem. Bootstrap techniques were used to generate more samples of data since the number of experimental data is reduced due to the costs and time durations of experimentations. We obtain in this way a better estimation of some drug parameters. Experiment results indicate that the proposed method is efficient.

## 13.1 Introduction

This article presents a dynamic neural network ensemble for modeling the situations that interfere in the process of designing drugs. By designing a drug, we mean to choose some variables of drug formulation, for obtaining optimal characteristics of drug [2, 3]. Our application is made on a particular class of drugs, namely retard drugs. We approach this problem with a bootstrap simulation which is suitable in some particular situations [1, 4].

The problem comes from the pharmaceutical research activity. It refers to a specific class of drugs that has delayed action called generically *retard drugs*. The pharmaceutical experimental situation leads to a mathematical optimization problem [12, 14, 15]. The pharmacist researcher must take into account several variables

**Table 13.1.** Sample data showing the *inputs* and *outputs*

| **Variables of formulation: Inputs** | | | | | | **Responses: Outputs** | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **ExpNo** | **X**$_1$ | **X**$_2$ | **X**$_3$ | **X**$_4$ | **X**$_5$ | **Y**$_1$ | **Y**$_2$ | **Y**$_3$ | **Y**$_4$ | **Y**$_5$ | **Y**$_6$ |
| 1 | 20 | 2 | 3 | 5 | 1 | 84.0 | 973.8 | 4.2 | 1.043 | 7.85 | 1.165 |
| 2 | 40 | 2 | 3 | 0 | 0 | 71.9 | 1150.0 | 1.6 | 1.016 | 8.2 | 2.264 |
| 3 | 20 | 8 | 3 | 0 | 1 | 92.5 | 1121.4 | 4.2 | 1.044 | 8.83 | 0.700 |
| 4 | 40 | 8 | 3 | 5 | 0 | 88.1 | 1200.0 | 3.7 | 1.038 | 8.87 | 1.205 |
| 5 | 20 | 2 | 9 | 5 | 0 | 99.2 | 910.0 | 5.8 | 1.061 | 8.3 | 1.914 |
| 6 | 40 | 2 | 9 | 0 | 1 | 68.2 | 985.1 | 4.1 | 1.043 | 7.9 | 2.550 |
| 7 | 20 | 8 | 9 | 0 | 0 | 99.1 | 1010.0 | 5.3 | 1.056 | 9.05 | 1.160 |
| 8 | 40 | 8 | 9 | 5 | 1 | 83.9 | 925.4 | 5.5 | 1.058 | 8.5 | 1.265 |
| 9 | 30 | 5 | 6 | 2.5 | 0.5 | 85.0 | 1055.8 | 3.8 | 1.036 | 8.3 | 1.535 |
| 10 | 30 | 5 | 6 | 2.5 | 0.5 | 81.2 | 1030.0 | 4.1 | 1.042 | 8.37 | 1.490 |
| 11 | 30 | 5 | 6 | 2.5 | 0.5 | 85.0 | 1060.0 | 4.1 | 1.042 | 8.4 | 1.535 |

of formulation of the drug such as: the speed of mixing turbine, the concentration of the binder, addition speed, the proportion of talc, the proportion of *lauril sulfate Na.*

We name these variables *inputs* and denoted as $X_i$, $i = \overline{1, n}$. For the problem considered, we have five inputs, $X_1, X_2, X_3, X_4$ *and* $X_5$. With those variables of formulation, for each combination taken into account, the researcher obtains a variant of drug with certain characteristics. For each obtained variant, some parameters are measured, called *responses* that characterizes the drug: charging performance, average diameter of the pill, Carr value, Hausner value, the flow time and the brittleness. We consider these responses as *outputs* denoted by $Y_j$, $j = \overline{1, m}$. For our problem, we have 6 outputs denoted by $Y_1, Y_2, Y_3, Y_4, Y_5$ and$Y_6$.

The costs of experimentations are high and it is necessary to devote a long time to determine all responses for each variant of drug. So, the research group realized only 11 experiments. With the formalization proposed above, we grouped the experimental data as depicted in Table 13.1.

The aim is to determine a combination of variables $(x_1, x_2, ..., x_5)$ such that the responses $(y_1, y_2, ..., y_6)$ are optimal. By optimal we mean that outputs respect some conditions and also by taking into account some restraints for outputs.

1. The first response, output $Y_1$, must be maximized, so the goal is to obtain a value as close as possible to 100 %.
2. Te second output $Y_2$ must not outrun some values determined by the fact it is a value representing tablet's diameter. So, the requirement is that y$_2$ $\in$ [800$\mu$m, 1000$\mu$m]. A value around the average 900$\mu$m is suitable.
3. The third output $Y_3$ has also an admissible interval for its value, [1, 20], but we must determine it as close as possible to 1.
4. The fourth response, output $Y_4$, has a narrower interval for its values, [1, 1.2], but it also has to be closest to 1.
5. The fifth output $Y_5$, representing a time quantity, must be as small as possible, but positive.
6. For the last output $Y_6$, the goal is to minimize it, with positive values, so the 0 value is considered desirable.

We search for the values of $X_i\ i = \overline{1,5}$ for which we obtain a drug formulation with optimal characteristics $Y_j\ j = \overline{1,6}$. Variables are chosen by the researcher from a continuous domain. Not all values are accepted. So we must consider domains of definition, real intervals, for each input variable. Accepted variation intervals for inputs, for our problem, are: $X_1$ ∈[0,50], $X_2$ ∈[1,8], $X_3$ ∈[3,9], $X_4$ ∈[0,5], and $X_5$ ∈[0,1].

## 13.2 Dynamic Ensemble of Neural Networks

Consider a population of *n* networks trained on a set A $=$ (x$_m$,y$_m$) of labeled instances of a binary classification problem. A simple approach to combining network outputs is to simply average them together [11, 13, 16]. The basic ensemble method (BEM) output is defined by:

$$f_{BEM} = \frac{1}{n}\sum_{i=1}^{n} f_i(x) \tag{13.1}$$

This approach can lead to improved performance, but does not take into account the fact that some networks may be more accurate than others. It has the advantage of being easy to understand and implement and can be shown not to increase the expected error. A generalization to the BEM method is to find weights for each output that minimizes the mean squared error (MSE) of the ensemble. The general ensemble model (GEM) is defined by:

$$f_{GEM} = \sum_{i=1}^{n} \alpha_i f_i(x) \tag{13.2}$$

where $\alpha_i$ is chosen to minimize the MSE with respect to the target function, *f* (estimated using the cross validation set), and sum to 1. Define the error, $\varepsilon_i$(x), of a network, $f_i$, as:

$$\varepsilon_i(x) = f(x) - f_i(x) \tag{13.3}$$

If the correlation matrix, C$_{ij}$ = $E[\varepsilon_i(x)\ e_j(x)]$, then the task is to find weights that minimize the following:

$$MSE_{[f_{GEM}]} = \sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j C_{ij} \tag{13.4}$$

It is shown that the optimal choice could be derived as follows:

$$\alpha_i = \frac{\sum\limits_{j=1}^{n} C_{ij}^{-1}}{\sum\limits_{k=1}^{n}\sum\limits_{j=1}^{n} C_{kj}^{-1}} \tag{13.5}$$

The performance of GEM depends on a reliable estimate of *C* and the fact that it is non-singular so that it can be easily inverted. In practice, errors are often highly correlated, thus, the rows of C are nearly linearly dependent so that inverting $C$ leads to significant round-off errors.

If the output of a neural network, $y = f_i$ (x), can be interpreted as the probability that an instance, $x$, is in a class, then, as $y$ approaches 1, we feel more certain that the instance is in the class. As $y$ approaches 0, we become more certain that the instance is not in the class. To quantify this notion; we define the certainty, $c(y)$, of a neural network output as:

$$c(y) = \begin{cases} y & if y \geq 0.5 \\ 1 - y & otherwise \end{cases} \tag{13.6}$$

The certainty rises for output y less than 0.5 as $y$ falls, and for outputs $y$ $\geq$0.5 as $y$ rises. It is possible to depict that one network output, $y_1$,is less certain than another, $y_2$, if $c(y_1)$ is less than $c(y_2)$. Note that the certainty behaves symmetrically with respect to positive and negative decisions; the certainty of an output of 0.1 is the same as that of an output of 0.9, but the decision they are certain about is different.

In a dynamic ensemble network (DEN), instead of choosing static weights derived from $f_i$ performance on a sample of the input space, the weights are adjusted proportional to the certainties of the respective network outputs [6, 7]. The dynamically averaged network (DAN) is defined as:

$$f_{DAN} = \sum_{i=1}^{n} w_i f_i(x) \tag{13.7}$$

where $w_i = \frac{c(f_i(x))}{\sum\limits_{i=1}^{n} c(f_i(x))}$ and $\sum\limits_{i=1}^{n} w_i = 1$

The weight vector is calculated each time the ensemble output is evaluated, to try to give the best decision for the particular instance under consideration, instead of statically choosing weights that give an optimal decision with respect to a cross validation set. Each network's contribution to the sum is proportional to its certainty. A value close to 0.5, for instance, would contribute very little to the sum while a very certain value of 0.99 (or 0.01) among many less certain values would dominate the sum. Each fully connected neural network in the ensemble is generated with random initial weights. Then, each neural network is trained partially with training data and tested with the validation data.

### 13.2.1 Artificial Neural Networks

Artificial neural networks (ANNs) were designed to mimic the characteristics of the biological neurons in the human brain and nervous system. An artificial neural network creates a model of neurons and the connections between them, and trains it to associate output neurons with input neurons. The network learns by adjusting the interconnections (called weights) between layers. When the network is adequately trained, it is able to generate relevant output for a set of input data. A valuable property of neural networks is that of generalization, whereby a trained neural network is able to provide a correct matching in the form of output data for a set of previously unseen input data.

## Multi-Layered Perceptron Networks (MLPN)

Typical MLPN is arranged in layers of neurons (nodes), where every neuron in a layer computes the sum of its inputs and passes this sum through a nonlinear function (an activation function) as its output. Each neuron has only one output, but this output is multiplied by a weighting factor if it is to be used as an input to another neuron (in a next higher layer). There are no connections among neurons in the same layer.

Activation functions for the hidden layers are required to introduce nonlinearity into the network. Without nonlinearity, hidden layers would not make networks more powerful. The training of a network by backpropagation (BP) involves three stages: the forward propagation of the input training pattern(s), the calculation and back-propagation of the associated error, and the adjustment of the weights. After training, application of the network involves only the computation of the feedforward phase.

Basically, BP is a gradient descent technique to minimize the error $E$ for a particular training pattern. For adjusting the weight $(w_k)$,in the batched mode variant the descent is based on the gradient $\bigtriangledown E(\frac{\delta E}{\delta w_k})$ for the total training set:

$$\triangle w_k(n) = -\epsilon \cdot \frac{\delta E}{\delta w_k} + \triangle w_k(n-1) \tag{13.8}$$

The gradient gives the direction of error $E$. The parameters $\varepsilon$ and $\alpha$ are the learning rate and momentum respectively. A good choice of both the parameters is required for training success and speed of the ANN.

In the conjugate gradient algorithm (CGA) a search is performed along conjugate directions, which produces generally faster convergence than steepest descent directions. A search is made along the conjugate gradient direction to determine the step size, which will minimize the performance function along that line. A line search is performed to determine the optimal distance to move along the current search direction. Then the next search direction is determined so that it is conjugate to previous search direction. The general procedure for determining the new search direction is to combine the new steepest descent direction with the previous search direction. An important feature of the CGA is that the minimization performed in one step is not partially undone by the next, as it is the case with gradient descent methods. An important drawback of CGA is the requirement of a line search, which is computationally expensive. Moller [5] introduced the scaled conjugate gradient algorithm (SCGA) as a way of avoiding the complicated line search procedure of conventional CGA. According to the SCGA, the Hessian matrix is approximated by:

$$E''(w_k)p_k = \frac{E'(w_k + \sigma_k p_k) - E'(w_k)}{\sigma_k} + \lambda_k p_k \tag{13.9}$$

where $E'$ and $E''$ are the first and second derivative information of global error function $E(w_k)$. The other terms $p_k$, $\sigma_k$ and $\lambda_k$ represent the weights, search direction, parameter controlling the change in weight for second derivative approximation and parameter for regulating the indefiniteness of the Hessian. In order to get a good quadratic approximation of $E$, a mechanism to raise and lower $\lambda_k$ is needed when the Hessian is positive definite. Detailed step-by-step description can be found in the literature [5]. We used the MLPN trained using SCGA.

## Elman Recurrent Neural Networks (ERNN)

ERNN, also known as partially recurrent neural network, are a subclass of recurrent networks [9]. They are multilayer perceptron networks augmented with one or more additional context layers storing output values of one of the layers delayed by one step and used for activating this or some other layer in the next time step. The ERNN has context units, which store delayed hidden layer values and present these as additional inputs to the network. The ERNN can learn sequences that cannot be learned with other recurrent neural network e.g. with Jordan recurrent neural network, since networks with only output memory cannot recall inputs that are not reflected in the output. Several training algorithms for calculation of error gradient in general recurrent networks exist. Usually, both hidden and output units have nonlinear activation functions. Note that external input at time $t$ does not influence the output of any unit until time $t + 1$. The network is thus a discrete dynamical system.

## Radial Basis Function Network (RBFN)

RBFN network consists of 3-layers: input layer, hidden layer, and output layer. The neurons in hidden layer are of local response to its input and known as RBF neurons, while the neurons of the output layer only sum their inputs and are called linear neurons. It is well known that neural network training can result in producing weights in undesirable local minima of the criterion function [10]. This problem is particularly serious in recurrent neural networks as well as for MLPN with highly nonlinear activation functions, because of their highly nonlinear structure, and it gets worse as the network size increases. This difficulty has motivated many researchers to search for a structure where the output dependence on network weights is less nonlinear. The RBFN has a linear dependence on the output layer weights, and the nonlinearity is introduced only by the cost function for training, which helps to address the problem of local minima. There are two basic methods to train an RBFN in the context of neural networks. One is to jointly optimize all parameters of the network similarly to the training of the MLPN. This method usually results in good quality of approximation but also has some drawbacks such as a large amount of computation and a large number of adjustable parameters. Another method is to divide the learning of an RBFN into two steps. The first step is to select all the centers $\mu$ in terms of an unsupervised clustering algorithm such as the $K$-means algorithm proposed by Linde et al., and choose the radii $\sigma$ by the $k$-nearest neighbor rule. The second step is to update the weights $B$ of the output layer, while keeping the $\mu$ and $\sigma$ fixed. The two-step algorithm has fast convergence rate and small computational burden. We used a two-step learning algorithm to speed up the learning process of the RBFN.

The selection of the centers and radii of RBF neurons can be done naturally in an unsupervised manner, which makes this structure intrinsically well suited for weather prediction. As a result, we adopt below a self-organized learning algorithm for selection of the centers and radii of the RBF in the hidden layer, and a stochastic gradient descent of the contrast function for updating the weights in the output layer. For the selection of the centers of the hidden units, we may use the standard $k$-means clustering algorithm. This algorithm classifies an input vector $x$ by assigning it the label most frequently represented among the $k$-nearest neighbor samples. Specifically, it places the centers of RBF neurons in only those regions of the input space, where significant data are present. Once the centers and radii are established, we can make use of the minimization of the contrast function to update the weights of the RBFN.

### Hopfield Model

This network is a single layer network with symmetric weight matrices in which the diagonal elements are all zero. The diagonal elements need not be zero, but we assume that is the case since the performance is improved when taken to be zero. Thus, for a Hopfield network with weight matrix $W$, $w_{ij} = w_{ji}$ and $w_{ii} = 0$ for all *i, j = 1, 2, . . . , n*. Inputs are applied simultaneously to all neurons, which then output to each other and the process continues until a stable state is reached, which represents the network output. The feedback loops involve the use of particular branches composed of unit-delay elements (denoted by $z^{-1}$), which result in a nonlinear dynamical behavior by virtue of the nonlinear nature of the neurons.

## 13.2.2 Ensemble of Neural Networks

Figure 13.1 represents the architecture of the ensemble neural network [13]. The ensemble is formulated using combinations of MLPN, ERNN, RBFN and HFM architectures using the dynamic ensemble network strategy discussed above.

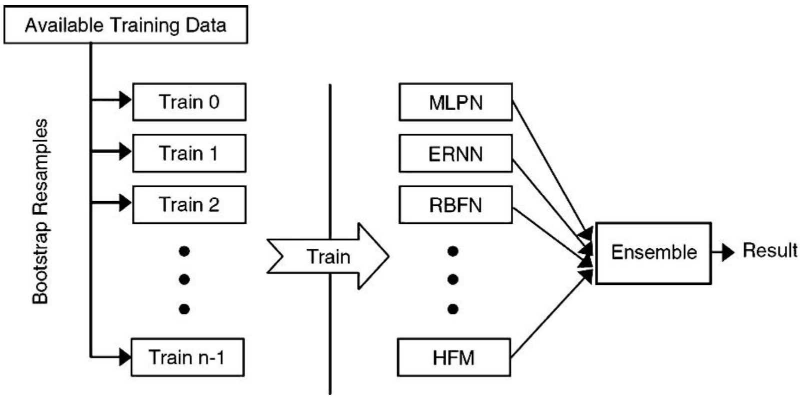

**Fig 13.1.** Dynamic ensemble architecture

## 13.3 Experiment Setup and Results

In this kind of pharmaceutical drug design problem, the researcher has to deal with two major aspects of the experimentation:

1. The relation between inputs and outputs is unknown, so we must find a way to make the best approximation
2. Data are difficult to obtain due to the restraints of costs and time. So, a method to simulate similar data is desirable

### 13.3.1 Bootstrap Re-sampling of Data

If the proposed solution is not acceptable or the system developed is not compatible and the process of evaluating an optimal solution is not convergent, we can conclude that the sample of data is too poor and the regression functions are not suitable to describe the links between variables. We resort, in this case, to a resampling method that allow us to manage uncertainty. The aim is to improve our sample of data with pseudo data and to evaluate some statistical parameters. We use a bootstrap method of resampling data. We group each input variable of formulation $X_i$, $i = \overline{1,5}$ with each output $Y_j$, $j = \overline{1,6}$. We obtain 30 vectors with bivariate data $(X_i,\ Y_j)$. For those vectors we apply a bivariate bootstrap resampling procedure. Finally, among the bootstrap simulated sets of X$_i$ variables we select a combination of inputs, a set of values $x_i, i = \overline{1,5}$, that correspond to best situated values of $Y_j$. We mean by this to observe the resampling process of $(X_i, Y_j)$ and select a proper combination of y$_j$, $j = \overline{1,6}$. In our application a such a combination is

$$(y_1 = 99.1, y_2 = 910, y_3 = 1.6, y_4 = 1.036, y_5 = 7.85, y_6 = 0.7)$$

We look in the bivariate bootstrap resampling process and extract the input combination $(x_1, x_2, x_3, x_4, x_5)$ for which we obtain the above combination of $y_j, j = \overline{1,6}$.

From other point of view, such a resampling method is useful to obtain new data and to improve the function approximation of the dependence $y = f(X)$. So, we can say that no matter what method we choose to approach such a problem, resampling bootstrap methods are desirable to improve the accuracy of data sets.

The experimental system consists of two stages: modeling the different neural network models (and constructing the ensemble) and performance evaluation. 70% of the data was used to train the different network models and the ensemble and 30% for testing purposes. Experiments were repeated three times and the worst errors are reported. The test data is then passed through the trained models to evaluate the learning efficiency of the considered models.

All the data were transformed into values between -1 and 1. The main goal of this scaling, in combination with weight initialization, is to allow the squashed activity function to work at least at the beginning of the learning phase. Thus, the gradient, which is a function of the derivative of the non-linearity, will always be different from zero. At the end of each algorithm, the outputs were re-scaled into the original data format for achieving the desired result.

**Table 13.2.** Test results and performance comparison of drug design system

| **Output** | **$Y_1$** | **$Y_2$** | **$Y_3$** | **$Y_4$** | **$Y_5$** | **$Y_6$** |
|---|---|---|---|---|---|---|
| **MLPN** | | | | | | |
| **RMSE** | 0.0487 | 0.039 | 0.043 | 0.040 | 0.038 | 0.039 |
| **CC** | 0.979 | 0.968 | 0.967 | 0.956 | 0.966 | 0.975 |
| **RBFN** | | | | | | |
| **RMSE** | 0.0322 | 0.0391 | 0.0390 | 0.0421 | 0.0412 | 0.0381 |
| **CC** | 0.972 | 0.981 | 0.972 | 0.964 | 0.956 | 0.973 |
| **ERNN** | | | | | | |
| **RMSE** | 0.0532 | 0.0456 | 0.0489 | 0.0534 | 0.0456 | 0.0453 |
| **CC** | 0.954 | 0.956 | 0.966 | 0.945 | 0.946 | 0.947 |
| **HFM** | | | | | | |
| **RMSE** | 0.0675 | 0.0756 | 0.0453 | 0.0563 | 0.0645 | 0.0756 |
| **CC** | 0.923 | 0.091 | 0.0967 | 0.0961 | 0.0933 | 0.0921 |
| **Dynamic Ensemble Network** | | | | | | |
| **RMSE** | 0.0192 | 0.0223 | 0.0213 | 0.0310 | 0.0202 | 0.0276 |
| **CC** | 0.988 | 0.991 | 0.987 | 0.976 | 0.978 | 0.983 |

The configuration of the neural network depends highly on the problem. To decide on the architectures of the MLPN, ERNN and HFM, a trial and error approach was used. Networks were trained for a fixed number of epochs, and the error gradient was observed over these epochs. Performance of the MLPN, ERNN and HFM networks were evaluated by increasing or decreasing the number of hidden nodes. The activation functions for the MLPN and ERNN models were chosen to be log-sigmoid and hyperbolic-tangent-sigmoid for hidden units, respectively, and linear for the output units.

Since there is no exact rule for fixing the number of hidden neurons and hidden layers to avoid under-fitting or over-fitting in the MLPN and ERNN networks, therefore, the RBFN model is investigated to address this difficulty. In RBFN, the Gaussian activation function was chosen for the hidden units, and linear for the output units.

The obtained results indicate that satisfactory accuracy has been achieved using the MLPN, ERNN, HFM and RBFN models. The performance is evaluated using root mean squared error (RMSE) and correlation coefficient (CC).

Compared to the HFM performance, MLPN exhibited lower errors. It is capable of modeling the problem considered better than HFM. However, the learning process of the MLPN algorithm is time-consuming and its performance is heavily dependent on the network learning parameters. The ERNN model, compared to MLPN, could efficiently capture the dynamics of the model, resulting in a more compact and natural representation of the temporal information contained in the data. The RMSE of the ERNN model was much lower than that of the HFM method. The RBFN performed well in terms of accuracy. Since RBFN has unsupervised learning characteristics and a modular network structure, these properties make it more effective for fast and robust problem modeling. It is indicated that the HFM model overestimated most of the predicted values. Overall, the performance of HFM is reasonable. However, compared to the other models, it is less accurate for the drug design problem.

The optimal network is the one that has a lower error and the highest correlation coefficient. The experimental comparisons of the MLPN, ERNN, RBFN and HFM methods pointed out that no single algorithm can be regarded as the best to model all variables. Thus, the use of ensemble of neural networks as an alternative approach is considered and the empirical performance clearly illustrates the importance of the approach. Dynamic ensemble-averaging tends to cancel out the noise part as it varies among the different ensemble members, and tends to retain the fitting of the regularities of the data.

## 13.4 Conclusions

This article introduced a dynamic ensemble neural network model for a pharmaceutical drug design problem. Test results reveal that the proposed connectionist models are capable of modeling all the outputs accurately. Compared to the different artificial neural network approaches, the dynamic ensemble model performed better in terms of RMSE and correlation coefficient values.

Performance could have been improved by providing more training data. The most important achievement of this result is that it gives to the researcher a new starting point of experimentation in stead of making other 20 - 30 experiments and to arrive to the same conclusion as the ensemble model recommends.

## Acknowledgements

Authors would also like to thank the colleagues of the Department of Maxillofacial Surgery, University of Medicine and Pharmacy, Iuliu Hatieganu Cluj-Napoca, for the initial contributions of this research.

## References

1. Abdelhak M. Zoubir, D. Robert Iskander, (1998), *Bootstrap MATLAB Toolbox*, Software Reference Manual.
2. Remus Câmpean, A. Prodan, (2003), *Biomatematică – aplicatii în Excel*, Editura Medicală Universitară "Iuliu Hatieganu", Cluj-Napoca, ISBN: 973-693-016-5.
3. Remus Câmpean, Augustin Prodan, (2003) *A Rating Model Applied for Designing Drugs*, Proceedings of the 12-th IASTED International Conference on Applied Simulation and Modelling, Marbella, Spain, pg 557-561, ACTA press, ISBN: 0-88986-384-9, ISSN: 1021-8181
4. T. Hesterberg, S. Monaghan, D. S. Moore, A. Clipson, R. Epstein, (1993), *Bootstrap Methods and Permutation Tests*, W. H. Freeman and Company, New York, 2003

5. Moller, A. F., A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning, Neural Networks. 6:525-533.
6. Hansen LK, Salamon P. (1990), Neural network ensembles. IEEE Transactions on Pattern Analysis 12(10):993-1001.
7. Jimenez D, Walsh N (1998) Dynamically weighted ensemble neural networks for classi.cation. In: Proceedings of the international joint conference on neural networks (IJCNN98), Anchorage, Alaska, May 1998, pp 753-756.
8. Sharkey AJC (1999) Combining artificial neural nets: ensemble and modular multi-net systems. Springer, Berlin Heidelberg New York.
9. Elman J.L. (1991), Distributed representations, simple recurrent networks and grammatical structure, Machine Learning, Vol. 7, No. 2/3, pp. 195-226.
10. Orr, M. J.(1995 ), "Regularization in the selection of radial basis function centers", Neural Computation, Vol. 7, No. 3, pp. 606-623.
11. Maqsood I, Khan MR and Abraham, A. (2004), Neural Network Ensemble Method for Weather Forecasting, Neural Computing & Applications, Springer Verlag London Ltd., Volume 13, No. 2, pp. 112-122.
12. Abraham, A., Grosan C. and Tigan S. (2007), Ensemble of Hybrid Neural Network Learning Approaches for Designing Pharmaceutical Drugs, Neural Computing & Applications, Springer Verlag London Ltd., Volume 16, No. 3. pp. 307-316.
13. Maqsood I and Abraham, A. (2007), Weather Analysis Using an Ensemble of Connectionist Learning Paradigms, Applied Soft Computing Journal, Elsevier Science, Volume 7, Issue 3, pp. 995-1004.
14. Grosan C., Abraham, A. and Tigan S.(2006), Engineering Drug Design Using a Multi-Input Multi-Output Neuro-Fuzzy System, 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'06), Timisoara, Romania, IEEE CS Press, pp. 365-371.
15. Grosan C., Abraham, A., Tigan S., Chang TG and Kim, DH (2006), Evolving Neural Networks for Pharmaceutical Research, International Conference on Hybrid Information Technology (ICHIT'06), IEEE Press, Korea, pp. 13-19.
16. Maqsood I, Khan MR and Abraham, A. (2004), Neural Network Ensemble Method for Weather Forecasting, Neural Computing and Applications, Springer Verlag London Ltd., Volume 13, No. 2, pp. 112-122.