SUPPLEMENTARY INFORMATION

Estimation of ternary liquid-liquid equilibria for arene/alkane/ionic liquid mixtures using neural networks

José S. Torrecilla [a,*], Maggel Deetlefs [b], Kenneth R. Seddon [b], Francisco Rodríguez [a]

[a] Department of Chemical Engineering, Faculty of Chemistry, Complutense University of Madrid, 28040-Madrid, Spain.
[b] The QUILL Research Centre, The Queen's University of Belfast, Belfast, BT9 5AG, UK.

1. Neural Network Models

In the current work, three types of neural network models (Radial Basis Network Fewer Neurons, RadialFN; Radial Basis Network Exact Fit, RadialEF and Multilayer Perceptron, MLP) were used to correlate the experimental LLE data and to estimate the toluene/*heptane* selectivity values. [23, 32] Descriptions of the three neural networks employed in the current work are given below.

1.1. Radial Basis model

The radial basis model consists of three layers: the input, hidden radial basis and output linear. The input layer has no calculation power and serves as an input distributor to the hidden radial basis layer. The net input to the hidden radial basis neuron is the vector distance between its weight vector (self-adjustable parameter of the net), *w,* and the input vector, *p*, multiplied by the bias. The two last layers have biases (Figure 1a). The transfer function of radial basis neurons is a Gaussian function, eq. (1). The radial basis function has a

* Corresponding author. Tel.: +34 394 42 40; Fax: +34 394 42 43. E-mail address: jstorre@quim.ucm.es (José S. Torrecilla).

maximum of 1 when its input is 0. As the distance between *w* and *p* decreases, the output increases. The bias allows the sensitivity of the radial basis neuron to be adjusted. The operation of the output layer is a linear combination of the radial basis units according to eq. (2).[23]

$$G_j(x) = \frac{1}{e^{x^2}} \qquad (1)$$

$$y_k(x) = \sum_{j}^{n_h} w_{jk} \cdot G_j(x) + w_k \qquad (2)$$

In eqs. (1) and (2), y_k is the k^{th} output unit for the input vector x, n_h is the number of hidden radial basis units, w_{jk} is the weight between the j^{th} hidden and the k^{th} output neurons, G_j is the notation for the output of the j^{th} radial basis unit, and w_k is the bias. Two different radial basis models *viz.* RadialFN and RadialEF were used in this work.

1.1.1 Radial basis networks fewer neurons. Initially, the hidden radial basis layer has no neurons and the RadialFN algorithm adds neurons to the hidden layer of a radial basis network until it meets the specified mean squared error goal (performance error). It depends on a matrix of input vectors, a matrix of target class vectors, a mean squared error goal, maximum number of neurons and a spread of radial basis functions (spread constant). Too large a spread means a lot of hidden neurons will be required to fit a fast-changing function. Too small a spread means many hidden neurons will be required to fit a smooth function, and the network may not generalize well.

The following steps are repeated until the network's mean squared error falls below performance error: (*i*) the network is simulated; (*ii*) the input vector with the greatest

error is found; (*iii*) a hidden radial basis neuron is added with weights equal to that vector (initially the radial basis layer has no neurons); (*iv*) the linear layer weights are redesigned to minimize error and reach the performance error value.

1.1.2 Radial basis networks exact fit. The RadialEF algorithm very quickly designs a radial basis network with zero error on the design vectors, *i.e.* in this model the performance error is equal to zero. It depends on a matrix of input vectors, a matrix of target class vectors and a spread of radial basis functions (spread constant). The RadialEF algorithm returns a new exact radial basis network.

1.2. Multilayer perceptron model

The multilayer perceptron model (MLP) consists of several neurons arranged in three layers: input, hidden, and output layers (Figure 1b). The input layer is used to input data into the MLP; the nonlinear calculations are carried out in the other layers. The total calculation process of the neural network can be classified in forward and back-propagation stages. These two calculation processes are explained below.

Forward calculation process (estimation process). The calculation process in each neuron of the hidden and output layers (represented by subscripts j and k, respectively) consists of activation and transfer functions. The activation function, eq. (3), means that the input data to each neuron (Figure 1b) are multiplied by a self-adjustable parameter, w_{ij} or w_{jk} (hidden or output layers, respectively), called weights; the result, x_k, is fed into a transfer function. The Sigmoid, Hyperbolical Tangent or Linear functions are the most commonly used as transfer functions. Given that every variable used in this work and the sigmoid function are

bounded between 0 and 1, the transfer function selected was the sigmoid transfer function, eq. (4). In the output layer, the calculated value, y_k, is the output of the considered neuron.

$$x_k = \sum_{j=1} w_{jk} \cdot y_j \qquad (3)$$

$$y_k = f(x_k) = \left(\frac{1}{1 + e^{-x_k}} \right) \qquad (4)$$

In eqs. (3) and (4), y_j and y_k represent the output of hidden neurons (j) and output (k) neurons, w_{jk} represents the weight between the j^{th} hidden and the k^{th} output neurons (Figure 1b).

Back-propagation calculation process (learning process): In this stage, the weights are updated as a function of mean square error (MSE), eq. (5), to improve the predictive capacity of the multilayer perceptron model. Specifically, in this stage, a training function uses this MSE to update the weights. The training functions are mathematical procedures used to automatically adjust all the MLP's weights and bias of a given network. Therefore, one of the most important parts of the MLP model is the selection of an adequate training function. The training function selected must be able to prevent the overfitting (violation of Occam's razor) and overtraining,[33] especially, when few learning data are used. The overtraining problem refers to the fact that the network only memorizes the learning set and loses its ability to generalize. With few learning data, the Bayesian Regularization back-propagation training function was selected to carry out the learning process because its generalization power is higher than other training functions (*e.g.* Gradient descent back-propagation, *quasi*-Newton back-propagation, *etc*.).[23] Bayesian regularisation minimises the prediction error by using a

linear combination of squared errors and weights, eq. (7). It then determines the correct combination so as to produce a network that generalizes new input data well inside the learning data range.

$$MSE = \frac{1}{N}\sum_{k}^{N}(r_k - y_k)^2 \qquad (5)$$

$$MSW = \frac{1}{N}\left[\sum_{i}^{N}\sum_{j} w_{ij}^{\ 2} + \sum_{j}^{N}\sum_{k} w_{jk}^{\ 2}\right] \qquad (6)$$

$$msereg = \gamma \cdot MSE + (1-\gamma)\cdot MSW \qquad (7)$$

In eqs. (5), (6) and (7), *N, y_k, r_k, w, i ,j and k* are the number of observations, neural network model estimation, real value, weight value between layers are described in the subscripts, input, hidden and output layers, respectively. In eq. (7), γ is the performance ratio, which was set to 0.5,[23] giving equal weight to the mean square errors and the mean square weights.

The Bayesian Regularization training function updates the weight and bias values according to Levenberg-Marquardt optimization. The Jacobian *jX* of performance with respect to the weight and bias variables *X* is calculated in the back-propagation calculation process.

$$jj = jX \cdot jX \qquad (8)$$

$$je = jX \cdot e \qquad (9)$$

$$dX = -\frac{jj + I\cdot\mu}{je} \qquad (10)$$

In eqs. (9) and (10), ***e*** is a vector of network errors and I is the identity matrix. The Bayesian Regularisation training functions parameters are learning coefficient (μ), learning coefficient decrease (μ_d) and learning coefficient increase (μ_i). μ is related with the rate to find the relative minimum error of the system to be modeled. Depending on the performance value, described by eq. (7), μ value decreases or increases, as a function of μ_d or μ_i values, respectively.

1.3. Learning and verification sample

The experimental data used in this work are shown in Figure 1 (main manuscript).[4] Every neural network model was designed to estimate tie lines in ternary mixtures. In Figure 2, the tie line (AB) is described by three molar fractions for each mixture (A and B), where A and B compositions represent two different phases. Because every sum of three molar fractions is equal to unity, each composition (A or B) is perfectly described by only two values (*e.g.* $X_{toluene}$, $X_{heptane}$, X_{IL}=1-$X_{toluene}$+$X_{heptane}$). Therefore, two pairs of molar fractions in two different phases ($X_{toluene}$, $X_{heptane}$, or X_{IL} in heptane rich phase and $X_{toluene}$, $X_{heptane}$, or X_{IL} in ionic liquid rich phase) are necessary to describe a tie line. These four molar fractions were classified in two groups, *viz.* dependent and independent variables, which in turn are composed of one and three molar fractions, respectively. To distribute the four above mentioned variables into two groups, all possible combinations were tested. Every combination was checked by a multiple regression analysis. In every system, the combination with the highest correlation coefficient was selected for further use (Table 1).

Since a mathematical relationship exists between the dependent and independent variables, the topology of every neural network consists of an input neuron (dependent variable), the number of hidden neurons is decided during the learning process and three output neurons (independent variables). Therefore, the learning and verification samples consist of these four variables, which are taken from the same source.[4] The only difference between the verification and learning samples is that the latter is composed of 80% of data and the former of the remaining 20%. Taking into account that every datum of the verification sample should be interpolated within learning range, the data were randomly distributed in both samples.

1.4. Optimisation process of the neural network models

1.4.1. Optimisation process of the radial basis models

The radial basis models *viz*. RadialFN and RadialEF were optimised in different ways. The spread constant and performance error values were calculated in the RadialFN neural network model optimization. To study the relationship between these parameters and optimise their values a multilevel factorial experimental design involving five levels for every experimental factor was carried out. The factors analyzed were spread constant (2 to $1 \cdot 10^{-3}$) and performance error (0.1 to $1 \cdot 10^{-4}$).[23] The response variables were the mean prediction error (MPE), eq. (11), correlation coefficient (R^2) (predicted *vs.* experimental values) and the root mean square deviation (RMSD), eq. (12).

$$MPE = \frac{1}{N}\sum_{k}|r_K - y_K| \qquad (11)$$

$$RMSD = \sqrt{\frac{\sum_{i}\sum_{l}\sum_{m}(x_{ilm}^{\exp} - x_{ilm}^{calc})^2}{6t}} \qquad (12)$$

In eqs. (11) and (12), N, y_k, r_k, x, i ,l ,m and t are the number of observations, neural network model estimation, real value, mole fraction, designation for the component, phase, tie line and number of tie lines, respectively.[4] The design was analyzed taking into account that the estimations should be carried out with the lowest MPE and RMSD values possible. In contrast, In the RadialEF model optimization, the spread constant was optimised. It was carried out by testing different spread constant values between 0.001 and 15.[23]

1.4.2. Optimization process of the MLP model

The number of neurons in the hidden layer was optimised between 5 and 10 neurons. Given the learning sample size, and that the generalization performance improves when the network is minimized,[34] the lowest number of neurons to estimate the tie lines was selected. Six hidden neurons were selected because the MLP model with five hidden neurons estimated the tie lines with MPE higher than 2%.

A Box-Wilson Central Composite design 2^3 + star points experimental design was used to plan the experimental conditions and to optimise the values of the MLP's parameters. The experimental factors analyzed were μ *and* μ_d (between 1 and 0.001) and μ_i (between 2 and 100).[35] The responses were the MPE, R^2 (predicted *vs.* experimental values) and RMSD. The considerations taken into account in the experimental design analysis were that the

estimations should be carried out with the lowest mean prediction error and RMSD values, and the highest correlation coefficient values (real *vs.* estimated values).

1.5. Testing of the optimised neural network models

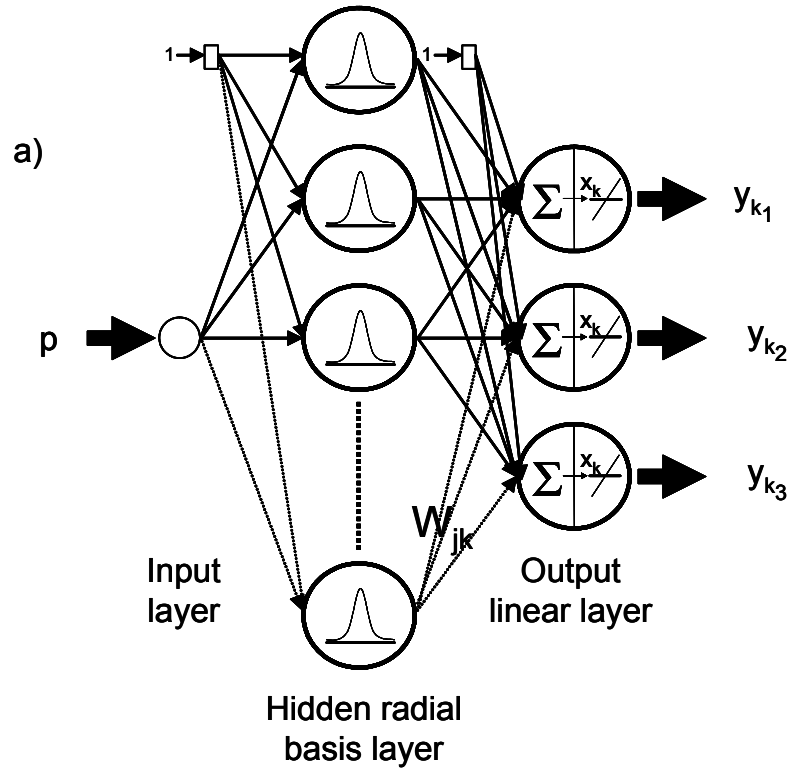
The verification sample was inputted into the optimised neural network models (radial basis network fewer neurons, radial basis network exact fit and multilayer perceptron) and their prediction and real values were then checked. The neural network evaluation consisted of applying statistical tests and calculating MPE, R^2 (predicted *vs*. experimental values) and RMSD values. To quantify the statistical difference between predicted and experimental databases, tests based on measures of central tendency (Kolmogorow-Smirnov test, Mann-Whitney-Wilcoxon test and Kruscal-Wallis test), on the dispersion (Kruscal-Wallis test, Cochran C test, Barllet's test and Levene test) and inferential parametric test for significance (F-Test and t-test) were applied. Each statistical test has an associated null hypothesis (predicted and experimental databases are statistically equal), the p-value, calculated as response of every test, is the probability that null hypothesis are fulfilled. The smaller the p-value, the more evidence exists against null hypotheses.[36]

2. Computational requirement

In every neural network used in this work, the input information required consisted of experimental literature data.[4] The number of neurons for the RadialFN, RadialEF and MLP neural network models was decided during the learning process. After the optimization process, in the verification stage, every neural network model performs a nominal computation to provide an output. No iterative computations are involved in providing an

output. In contrast, in the NRTL model, binary interaction parameters, pure component properties, and iterative computation is required to calculate an output. However, this thermodynamic method gives considerations about the physical model of the system. Therefore, the most adequate model depends on its final application. If a fast response (control process, on-line measurement, chemical engineering applications, *etc.*) is required or whenever the aspect of theoretical comprehensibility may become less important, neural network models are suitable alternatives.

Figure 1. Scheme of calculation (□ bias node); a) Radial basis network; b) Multilayer perceptron.



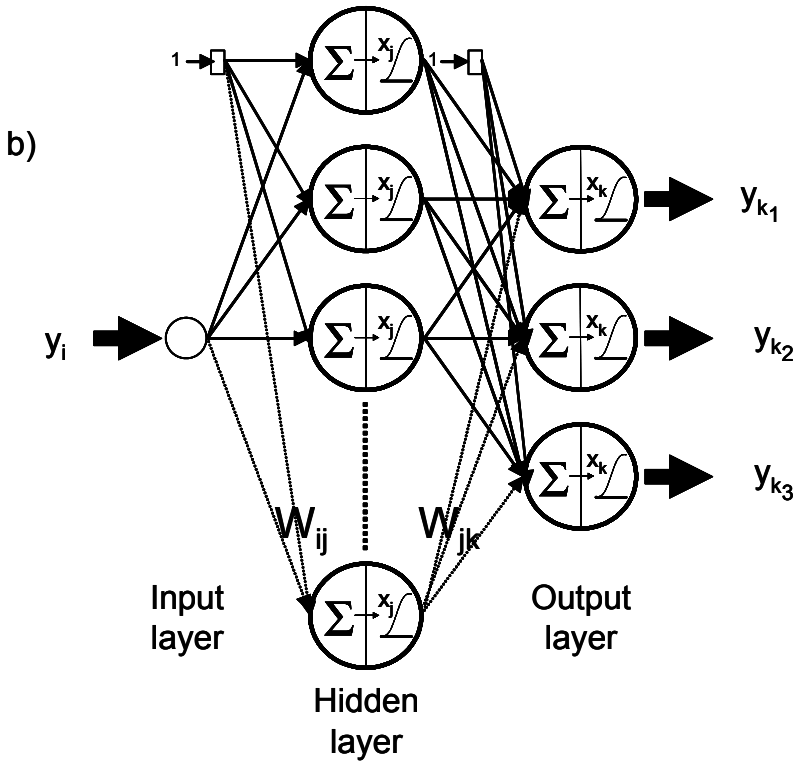


Figure 2. Generic tie line description (this plot is only used to explain the learning/verification data, it is not a realistic diagram).

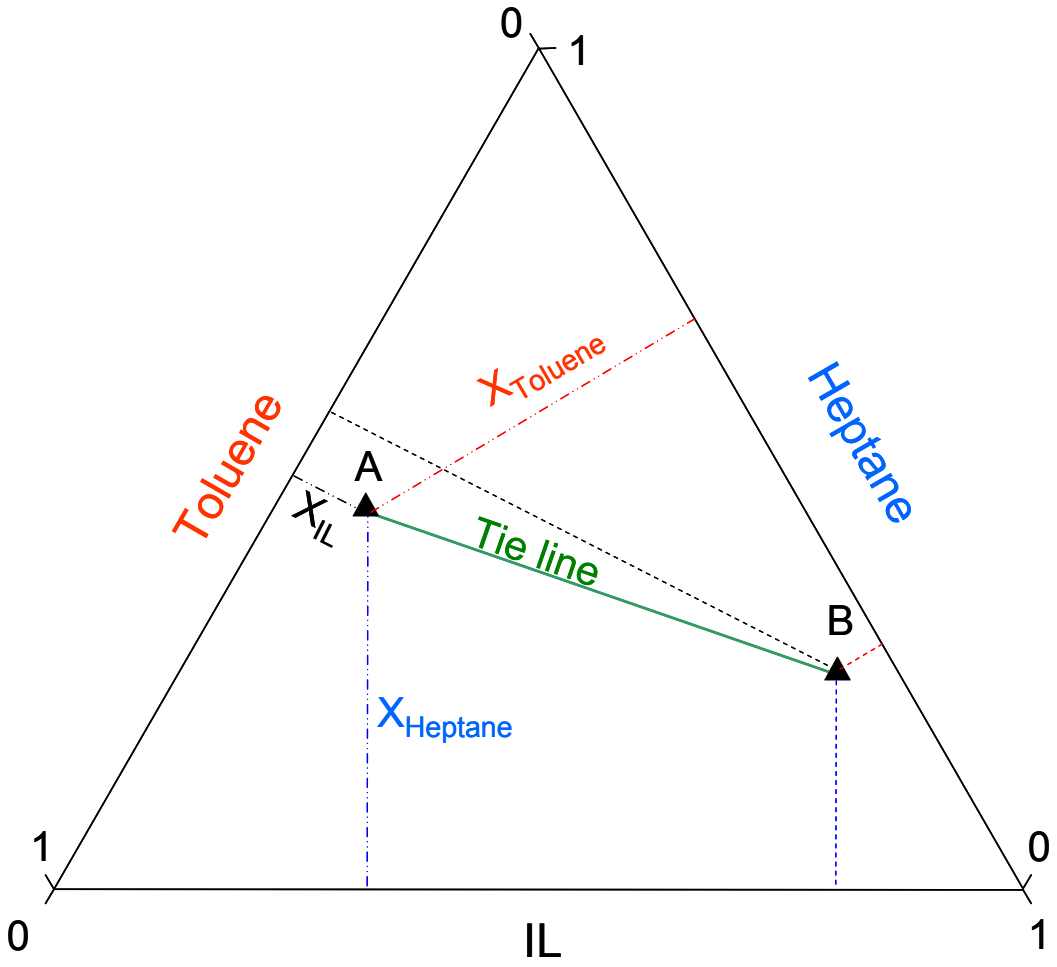


Table 1. Distribution of variables in independent and dependent variables group.

System	Independent Variables	Dependent Variables	Correlation Coefficient
1[a]	${}^{H}X[C_2mim][EtSO_4]$ ${}^{IL}X[C_2mim][EtSO_4]$ ${}^{IL}x_{heptane}$	${}^{H}X_{heptane}$	0.987
2[b]	${}^{H}X[C_2mim][MeSO_4]$ ${}^{IL}X[C_2mim][MeSO_4]$ ${}^{IL}X_{heptane}$	${}^{H}X_{heptane}$	0.990
3[c]	${}^{H}X[C_1mim][MeSO_4]$ ${}^{IL}X_{toluene}$ ${}^{IL}X_{heptane}$	${}^{H}X_{toluene}$	0.988
4[d]	${}^{H}X_{toluene}$ ${}^{IL}X_{toluene}$ ${}^{IL}X[C_1mim][MeSO_4]$	${}^{H}X_{heptane}$	0.999

[a] Toluene, heptane, $[C_2mim][EtSO_4]$ (T = 313.2 K, P = 0.1 MPa);
[b] Toluene, heptane, $[C_2mim][EtSO_4]$ (T = 348.2 K, P = 0.1 MPa);
[c] Toluene, heptane, $[C_1mim][MeSO_4]$ (T = 313.2 K, P = 0.1 MPa);
[d] Toluene, heptane, $[C_1mim][MeSO_4]$ (T = 348.2 K, P = 0.1 MPa).
IL Measured in the ionic liquid rich phase.
H Measured in the heptane rich phase.

References

4 G. W. Meindersma, A. J. G. Podt, and A. B. de Haan, *Fluid Phase Equilib.,* 2006, **247**, 158.

23 H.Demuth, M. Beale, and M. Hagan, in Neural Network Toolbox for Use with MATLAB® User's Guide. Version 5. Ninth printing Revised for Version 5.1 (Release 2007b); 2007 (online only).

32 D. M. Himmelblau, *Korean J. Chem. Eng.,* 2000, **17**, 373.

33 I. V. Tetko, D. J. Livingstone and A. I. Luik, *J. Chem. Inf. Comput. Sci.*, 1995, **35**, 826.

34 J. S. Torrecilla, M. L. Mena, P. Yáñez-Sedeño and J. García, *J. Food Eng.,* 2007, **81**, 544.

35 V. Vacic, Summary of the training functions in Matlab's NN toolbox. http://www.cs.ucr.edu/~vladimir/cs171/nn_summary.pdf, 2005.

36 J. S. Torrecilla, J. M. Aragón and M. C. Palancar, *Ind. Eng. Chem. Res.,* 2005, **44**, 8057.