Available online at www.sciencedirect.com
SCIENCE DIRECT®
Chemical Engineering and Processing 42 (2003) 715–721
Chemical Engineering and Processing
www.elsevier.com/locate/cep

Solving differential equations with unsupervised neural networks

Daniel R. Parisi [a,*], María C. Mariani [b], Miguel A. Laborde [a]

[a] Departamento de Ingeniería Química, Facultad de Ingeniería, Universidad de Buenos Aires, Pabellón de Industrias, Ciudad Universitaria (1428), Buenos Aires, Argentina

[b] Departamento de Matemática, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Pabellón I, Ciudad Universitaria (1428), Buenos Aires, Argentina

Received 26 April 2002; received in revised form 30 October 2002; accepted 30 October 2002

Abstract

A recent method for solving differential equations using feedforward neural networks was applied to a non-steady fixed bed non-catalytic solid–gas reactor. As neural networks have universal approximation capabilities, it is possible to postulate them as solutions for a given DE problem that defines an unsupervised error. The training was performed using genetic algorithms and the gradient descent method. The solution was found with uniform accuracy (MSE $\sim 10^{-9}$) and the trained neural network provides a compact expression for the analytical solution over the entire finite domain. The problem was also solved with a traditional numerical method. In this case, solution is known only over a discrete grid of points and its computational complexity grows rapidly with the size of the grid. Although solutions in both cases are identical, the neural networks approach to the DE problem is qualitatively better since, once the network is trained, it allows instantaneous evaluation of solution at any desired number of points spending negligible computing time and memory.

Keywords: Neural networks differential equations; Non-catalytic solid–gas reactor simulations

1. Introduction

The general method for solving differential equations with unsupervised feedforward neural networks was first introduced by van Milligen et al. [1]. They stated the general method and applied it to a magnetohydrodynamic plasma equilibrium problem. Other authors applied this method to other physical problems. Monterola and Saloma [2,3] solved the non-linear Schrödinger equation. Quito et al. [4] used neural networks for solving self-gravitating systems of N-bodies. As far as we know, the method has not been applied to solve chemical reactor problems.

Finding a neural network that approximates the solution of a given set of differential equations has many benefits compared with traditional numerical methods. First of all, the solution is continuous over all the domain of integration. In contrast, the numerical methods provide solutions only over discrete points; and the solution between these points must be interpolated. The computational complexity does not increase considerably with the number of sampling points and with the number of dimensions involved in the problem. Also the rounding-off error propagation of standard numerical methods does not affect the neural network solution.

The use of neural networks (NN) in the present work is totally different from that in a previous one [5], where a set of ODEs was solved first numerically and then two networks were trained using the solution points obtained. Here, the NN are proposed directly as solutions of the differential equations system, without knowing any solution point in advance, this means, the NN must be trained in an unsupervised manner.

Also, the present application of NN in chemical engineering is used from the point of view of a new mathematical method for solving general differential equations, which describe any particular system. This

* Corresponding author. Tel.: +54-11-4576-3240; fax: +54-11-4576-3241.

E-mail addresses: parisi@di.fcen.uba.ar (D.R. Parisi), mcmarian-@dm.uba.ar (M.v.C. Mariani), miguel@di.fcen.uba.ar (M.A. Laborde).

0255-2701/03/$ - see front matter
doi:10.1016/S0255-2701(02)00207-6

means that the chemical system is not considered as a black box, as it could be done in other supervised NN applications.

In the next section the general method for solving differential equations with NN is outlined. Then the simulation of an unsteady solid–gas reactor is solved as an application example. Also, the problem is solved using a traditional numerical method for validating and comparing.

2. The method

Any set of differential equations can be represented by the following expression

$$\boldsymbol{D}(f(z)) = 0 \tag{1}$$

where $\boldsymbol{D}$ is any non-linear, inhomogeneous differential operator and $f(z)$ is the solution that satisfies Eq. (1) and the appropriate boundary conditions.

Considering that a feedforward neural network is an universal function approximator [6–13], the goal is to find a neural network $f^*(z)$ which approximates $f(z)$ in the finite domain $z \in [a, b]^n$.

It is well known from neurocomputing sciences [14–18] that, in the case of one hidden layer, the functional form of component 's' of network's output f^* is given by

$$f_s^* = g\left(\sum_u w_{su} g\left(\sum_v w_{uv} x_v + b_u\right) + b_s\right) \tag{2}$$

where the w's are the network's adaptive coefficients (weights), the b's are the bias and g is a sigmoid activation function. Note that f^* is a continuous and derivable function of z, therefore the differential operator $\boldsymbol{D}$ can act on it.

In order to find an approximation of $f(z)$, it is natural to choose Eq. (1) plus the equations defining the boundary conditions as the performance function of the network. This error measure (E) must be evaluated in a finite number of points (P) into the integration domain $z^i \in [a, b]^n$.

$$E(w) = \frac{1}{P}\sum_i^P [\boldsymbol{D}(f^*(z^i))]^2 + \frac{1}{P_B}\sum_j^{P_B} [\boldsymbol{B}(f^*(z^j))]^2 \tag{3}$$

where $\boldsymbol{B}$ is a differential operator defining the boundary conditions. As E tends to zero, f^*tends to f and so the approximate solution for the differential equation system is found. As the error does not depend on target outputs (the function f is unknown a priori) the network is said to be trained in an unsupervised manner.

Training the neural network means to find the correct w's so that $E(w)$ is reduced to zero. In unsupervised learning, it is impossible to use the backpropagation algorithm because the error at each output unit is not available to the learning system. So, standard optimization techniques must be used. One of the simplest is the gradient descent method, the weights are initialized randomly and then, the following change rule is applied:

$$w_{kl}^{q+1} = w_{kl}^q - \eta\left(\frac{\partial E}{\partial w_{kl}^q}\right) \tag{4}$$

where η is the learning rate and q is the iteration step. There are other more efficient techniques as for example quasi-Newton and conjugate gradient.

Schaffer et al. [19] point out that 'genetic algorithms' can be most useful in finding weights in unsupervised learning tasks. There are other works dealing with genetic algorithms and neural networks [20–22]. For an introduction to this subject see Goldberg [23] or Mitchell [24]. The genetic method has the advantage that it searches the weight space without any gradient information of the error function ($E(w)$) whose derivatives can be more or less complicated depending on the particular operators $\boldsymbol{D}$ and $\boldsymbol{B}$ in the Eq. (3).

The ideal searching method could be a combination of both. First a global search using genetic algorithm. Then, a more exact position of minima can be obtained with gradient descent techniques.

It must be noted that the method for solving differential equations with neural networks is independent of the training method chosen, which can accelerate or slow down the speed of the training phase.

The method that allows solving differential equations with neural networks has several advantages compared with traditional numerical methods. The solution is analytic over the entire domain, which allows computing solution values for any input rapidly and gives a compact expression for that solution; computational complexity does not increase rapidly with the number of sampling points and there is no propagation of rounding-off errors affecting the accuracy of the solution. In contrast, standard methods provide solution values only at discrete locations of the solution space and the complexity increases rapidly with the number of sampling points.

Another important advantage is the following: If there are free parameters (as for example kinetic constants, diffusion coefficients, etc.) in the model, they could be treated as variables, extending the number of dimensions of the problem. Consequently, the function f^* (x, y, t, k_i, D_i, ...), will be a continuous function of the original variables and the free parameters. This simplifies the optimization of the model (comparing f^* versus experimental data) with respect to the free parameters. Since it would be not necessary to solve the DE problem (1) each time the free parameters change, as it would be the case, if the solution were obtained by any traditional numerical integration method.

The main difficulty of the method lies on the training phase, which may be time consuming, since unsupervised training is in general not simple.

3. Example: unsteady solid–gas reactor

The method described above is applied to a chemical reactor problem. Let simulate a fixed bed tubular reactor filled with porous pellets and two different reacting gases are fed at the reactor inlet ($y = 0$).

The chemical reactions involved are:

Reaction 1 $\quad A(g) + \alpha S(s) \rightarrow$ products

Reaction 2 $\quad B(g) + \alpha S(s) \rightarrow$ products

The extent of conversion of each gaseous reactant is defined instantaneously by the following equations

$$X_1 = C_A^0 - C_A, \tag{5}$$

$$X_2 = C_B^0 - C_B, \tag{6}$$

and in the solid phase

$$X_S = X_1 + X_2 = \frac{1}{\alpha}(C_S^0 - C_S) \equiv X_3 \tag{7}$$

where C is the concentration (mol/cm^3) and it can be related to the molar flow (F) using the following equation

$$F_n = C_n u A_R, \tag{8}$$

being u the gas velocity and A_R the crosssection of the tubular reactor.

The reacting system is assumed to be isothermal. Gases in the reactor follow a plug flow regime (the axial direction is the relevant spatial dimension). As the solid is fixed, there are no steady state conditions, so the time evolution must be considered.

Under these assumptions, mass balances in each phase, leads to a set of three coupled first-order partial differential equations that describes the reacting system:

$$\frac{\partial X_1}{\partial t} + u\frac{\partial X_1}{\partial y} + R_1(X_1,\ X_3) = 0 \tag{9}$$

$$\frac{\partial X_2}{\partial t} + u\frac{\partial X_2}{\partial y} + R_2(X_2,\ X_3) = 0 \tag{10}$$

$$\frac{\partial X_3}{\partial t} + R_1(X_1,\ X_3) + R_2(X_2,\ X_3) = 0 \tag{11}$$

where X_m is the extent of conversion of reactant $m = 1, 2$ or 3, corresponding to A, B or S, respectively; and R_n (< 0) is the reaction rate of the gaseous reactants $n = 1$, 2 (A and B).The boundary conditions for the system are

$$X_{1(y=0,\ t)} = 0 \quad \text{for } t \in [0,\ 1] \text{ at } y = 0 \tag{12}$$

$$X_{2(y=0,\ t)} = 0 \quad \text{for } t \in [0,\ 1] \text{ at } y = 0 \tag{13}$$

and the initial conditions are

$$X_{3(ut, y/u)} = 0 \quad \text{for } y \in [0,\ 1] \text{ at } t = y/u \tag{14}$$

$$\frac{\partial X_1}{\partial y} = \frac{-R_1(X_1,\ 0)}{u} \quad \text{for } y \in [0,\ 1] \text{ at } t = y/u, \tag{15}$$

$$\frac{\partial X_2}{\partial y} = \frac{-R_2(X_2,\ 0)}{u} \quad \text{for } y \in [0,\ 1] \text{ at } t = y/u, \tag{16}$$

Note that initial conditions (14–16) take into account that the problem is undefined before $t = y/u$, when the gases reach a given point y, at the beginning of the simulation.

In a direct application, it would be possible to solve this PDE problem using the NN method described above. Eqs. (9)–(11) would be the particular form of the general operator $\boldsymbol{D}$ in Eq. (1) and Eqs. (12)–(16) would define the particular form of the general boundary operator $\boldsymbol{B}$ in Eq. (3). If this were the case, X_1, X_2 and X_3 would be approximated by three different neural networks X_1^*, X_2^* and X_3^*.

However, an easier ODE problem can be obtained from the PDE problem (9)–(16) before applying neural networks. Consider the following transformation:

$$t' = t - y/u, \tag{17}$$

$$y' = y \tag{18}$$

The derivatives respect to the new variables result:

$$\frac{\partial(\ )}{\partial t} = \frac{\partial(\ )}{\partial t'}\frac{\partial t'}{\partial t} + \frac{\partial(\ )}{\partial y}\frac{\partial y'}{\partial t} = \frac{\partial(\ )}{\partial t'} \tag{19}$$

$$\frac{\partial(\ )}{\partial y} = \frac{\partial(\ )}{\partial t'}\frac{\partial t'}{\partial y} + \frac{\partial(\ )}{\partial y'}\frac{\partial y'}{\partial y} = -\frac{\partial(\ )}{\partial t'} + \frac{\partial(\ )}{\partial y'} \tag{20}$$

And the PDE system (9)–(11) can be re-written as

$$u\frac{\partial X_1}{\partial y'} + R_1(X_1,\ X_3) = 0 \tag{21}$$

$$u\frac{\partial X_2}{\partial y'} + R_2(X_2,\ X_3) = 0 \tag{22}$$

$$\frac{\partial X_3}{\partial t'} + R_1(X_1, X_3) + R_2(X_2, X_3) = 0 \tag{23}$$

with the boundary conditions:

$$X_{1(y'=0,\ t')} = 0 \tag{24}$$

$$X_{2(y'=0,\ t')} = 0 \tag{25}$$

$$X_{3(y',\ t'=0)} = 0 \tag{26}$$

Concerning the kinetic expression (R), the unreacted shrinking core model with first order reaction was chosen. The expression for the reaction rate per unit reactor volume, R_n is [25]

$$R_n = \frac{-n_p 4\pi r_c^2 C_n}{[1/k_n + r_c/D_n - r_c^2/r_o D_n]}, \tag{27}$$

where $n = 1$, 2 indicates the gaseous reactant n_p is the number of pellets per unit volume; C is the concentration of the gas outside the pellet (it is assumed that

external diffusional resistance is negligible); k is the kinetic constant for the surface reaction; D is the effective diffusion coefficient; r_0 is the external radius of the pellet and r_c is the radius of the unreacted core related to X_3 by

$$r_c = r_0\left(1 - \frac{\alpha X_3 \mathrm{M}}{\rho}\right)^{1/3}, \tag{28}$$

where M and ρ are the molecular weight and the density of the solid reactant, respectively.

For simplicity, it is assumed that in Eqs. (21) and (22), the variation of X_3 with respect to y' is negligible comparing with the X_1 and X_2 gradient. Physically, it means that, for a given instant t', the gaseous species will have more pronounced variations of concentration that the solid pellets along the reactor. This is a reasonable simplification if the difference of densities between solid and gaseous phases is taken into account. However in Eq. (23) the dependence of X_3 with respect to y' will be considered.

Under this assumption, Eqs. (21) and (22) with boundary conditions (24)–(25) can be solved analytically:

$$X_n = C_n^0\left(1 - \exp\left(-\frac{\beta(X_3)}{u}\right)y'\right), \tag{29}$$

for $n = 1, 2$ being $\beta = R_n/C_n$.

It remains X_3 as unknown and it must be solved from the single ODE Eq. (23) considering X_1 and X_2 as in Eq. (29) and y' as a parameter.

4. Results

In this section Eq. (23) is solved using the proposed NN method and a traditional Runge–Kutta method for comparison of the solutions obtained.

It is important to remark an essential difference between the two approaches. In the case of the traditional method the dimension y' must be discretized into y'_i, and for each y'_i Eq. (23) must be solved numerically. On the other hand, if the NN method is applied, the network can be trained over the (y', t') domain, providing continuous solution for $X_3(y', t')$, which is easier to include in the expressions for X_1 and X_2 given in Eq. (29), and this training must be performed only once.

The parameter values were $k_1 = 72.5$ cm/s; $k_2 = 33.6$ cm/s; $D_1 = D_2 = 0.03$ cm^2/s; $r_0 = 0.6$ cm; $M = 160$; $\rho = 3.5$ g/cm^3; $u = 100$ cm/s; $\alpha = 1/3$; $C_A^0 = 2.4011 \times 10^{-5}$ mol/cm^3; $C_B^0 = 1.8054 \times 10^{-5}$ mol/cm^3; $C_S^0 = 0.0219$ mol/cm^3; $A_R = 2.84$ cm^2.The activation function used in all the network units was:

$$g(x) = \frac{2}{(1 + \exp(-2x))} - 1. \tag{30}$$

A neural network approximation of X_3 with two units in the input layer (corresponding to y' and t'), five units in the hidden layer and one unit in the output layer, was trained to solve the ODE Eq. (23) with boundary condition Eq. (26).

The criteria for choosing the number of hidden layer, was to select the simplest network's architecture and the less number of units that can solve the problem. Trial and error beginning for one layer with few neurons show good network's capabilities. In general, if fewer hidden units are used better generalization may be attained. Besides, the training should be easier since fewer parameters have to be adjusted.

The unsupervised error function was defined as stated in Eq. (3) using the particular differential operators given by Eqs. (23) and (26). As the boundary terms $\boldsymbol{B}$ are smaller than the first term of the r.h.s. of Eq. (3) it must be multiplied by an arbitrary factor (in our case, 10 $(P+P_B)$), in order to balance the error components. The correct definition of the Error function is crucial to succeed with the training and the fact that it is an unsupervised problem makes the task more difficult.

The network was trained first using a genetic algorithm to get promising regions of the error landscape. Each individual has the weights and bias corresponding to the network ($5 \times 2+5+5 \times 1+1 = 21$ parameters, in the case of 5 hidden units). The mutation and crossover operators were defined as stated by Montana and Davis [21]. As the genetic algorithm has difficulties to find exact minimums, a second training phase follows using a gradient descent routine Eq. (4) with adaptative parameter η.

The training set was composed by $P+P_B = 110$ equidistant points into the range: $t'\varepsilon[0, 1]$ and $y'\varepsilon[0, 1]$. The mean squared error typically started around $E = 1$ and it reached $E = 2 \times 10^{-9}$. This error is computed without the arbitrary factor multiplying operator $\boldsymbol{B}$.

The network was tested with points that did not participate of the training. Fig. 1 shows the mean squared error (E) of the trained network as a function of the number of testing points.

It can be observed that the averaged error is stable for increasing number of testing points, which indicates that the solution found is a good approximation of the real solution not only for the training points but also over the whole domain of the problem.

The genetic algorithm takes around 500 generations and the gradient descent routine (4) takes approximately 2×10^4 iterations to converge.

The whole training phase may take several minutes in a PC. The particular method used for training the network is not unique and perhaps not the best for

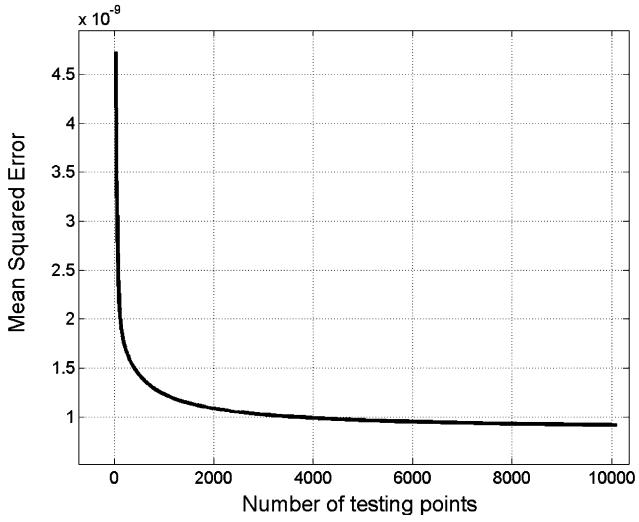

Fig. 1. Mean squared error of the trained NN vs. number of testing points into the integration domain.

unsupervised network, so more effort must be done in order to improved its efficiency.

However, the computing time requirement of the proposed NN method is not comparable with traditional numerical methods. Any numerical method gives solution points over a selected grid. The computing time, in this case, increases as the number of the grid points increases. So, computing the solution for an infinite number of points will need an infinite time. On the other hand, the NN find a continuous solution over the real domain in a finite time.

Once the NN X_3^* is trained, a continuous expression—with the form of Eq. (2)—that approximates X_3 is obtained, which can be easily replaced in the X_1 and X_2 expressions, and in this way, continuous solutions for the whole problem are found.

The present problem can also be solved by classical numerical methods.

In order to compare and to validate the results obtained by the neural network, the ODE problem (23) is solved numerically with an explicit Runge–Kutta method based on the Dormand–Prince pair [26]. As it was said before, the y' dimension must be discretized, and for each y'_i Eq. (23) must be integrated.

Using this method the absolute error is of the order of 10^{-5}, which is similar to the absolute error found by the NN. The computing time needed for integrate the problem in a small grid of point is a few seconds, but if the discretization of the domain contains about 1000 points for each variable, the computing time of the 1000 Runge–Kutta (one for each y'_i) resolutions is similar to that of the NN training.

Figs. 2–4 show the superposition of the continuous solution—obtained by the neural network approximation X_3^* and by Eq. (29) for X_1 (X_3^*) and X_2 (X_3^*)—over 256 points covering the integration domain and the

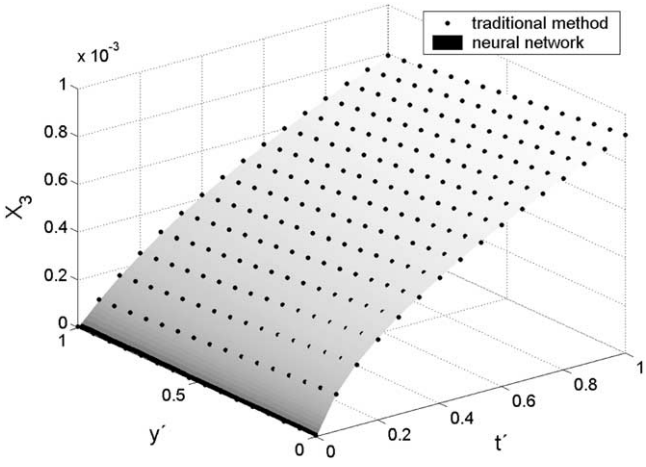

Fig. 2. Neural network an traditional numerical method solutions of Eq. (23) for X_3.

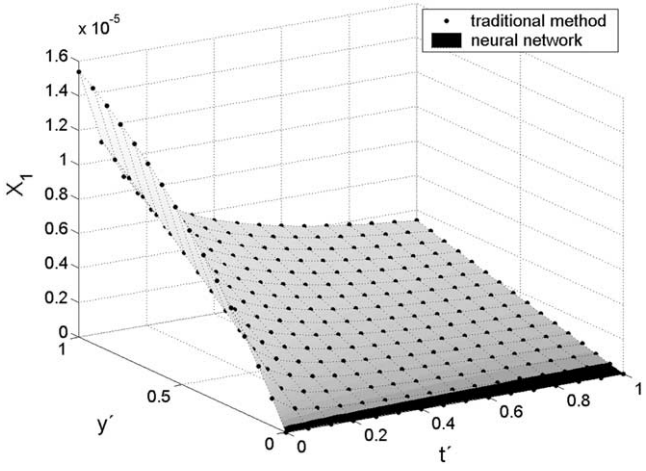

Fig. 3. X_1 solution (Eq. (29)) using X_3 found by the two methods.

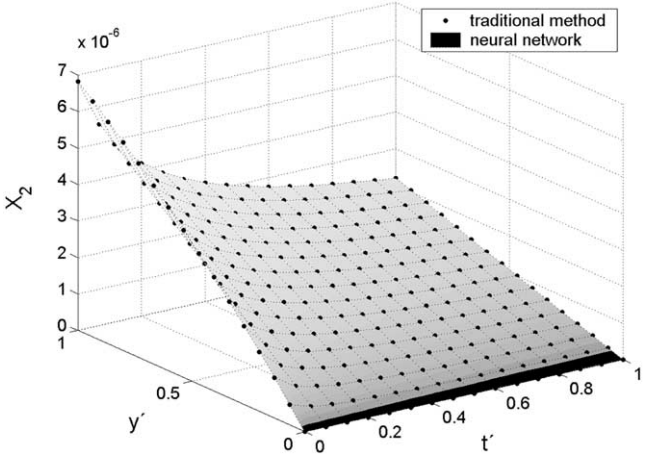

Fig. 4. X_2 solution (Eq. (29)) using X_3 found by the two methods.

equivalent solution points obtained from the traditional numerical method.

It can be seen that, for the 256 points plotted, there are not appreciable differences between the solutions found by both methods.

It must be noted that the trained neural network provides a compact expression (Eq. (2)) for X_3 (and consequently for X_1 and X_2), which allows evaluating any number of points spending a negligible computing time and memory requirements. Also, the NN has the flexibility to manage parameters as variables (like y' in the present example). These characteristics would be especially useful when many simulations must be carried out for optimization or control problems.

The same task could not be done so easily by a traditional numerical integration since it would be necessary to completely solve the problem each time the parameters are changed and always the solution will have a non-compact form (a list of points).

5. Conclusion

A new and powerful application of unsupervised neural networks for solving differential equations was implemented in a chemical engineering problem: a non-steady state fixed bed solid–gas reactor.

In order to validate the neural network solution, comparison with a traditional numerical method (Runge–Kutta) was made. The solutions obtained in both cases look identical.

The neural network was trained with a finite set of input points (y^i, t^i) but it provides analytical solution at any point (y, t) into the real integration domain, keeping the same mean squared error level.

In contrast, the Runge–Kutta method gives solution points only over a predefined grid. Its PC time and memory requirements can not be compared to that of NN since they grow rapidly with the number of points in the grid.

Finally, the neural network approximation of the solution is a compact mathematical expression (Eq. (2)), which can include any number of parameters as variables, and consequently, it is more adequate to be used in optimization or control problems.

Acknowledgements

Authors would like to acknowledge to the University of Buenos Aires for financial support and to José Comas for text revision.

Appendix A: Nomenclature

A gaseous reactant
$\boldsymbol{B}$ general differential operator defining the boundary conditions
B gaseous reactant
b neuron bias
C concentration of the gas outside the pellet (mol/cm^3)
D effective diffusion coefficient (cm^2/s)
$\boldsymbol{D}$ general differential operator
E unsupervised NN error function
f exact solution of $\boldsymbol{D}$
f^* neural network approximation of f
g neuron sigmoid activation function
k kinetic constant of the surface reaction (cm/s)
M molecular weight
n_p number of pellets per unit volume (1/cm^3)
P number of sampling points
R reaction rate (mol/cm^3 s)
r_0 external radius of the pellet (cm)
r_c radius of the unreacted core (cm)
S solid reactant
t time (s)
u gas velocity (cm/s)
w network's adaptive coefficients (weights)
X extent of conversion of reactant (mol/cm^3)
X^* neural network approximation of X (mol/cm^3)
x dummy variable
y space variable inside the reactor (cm)
z general vector variable of f

Subscripts
B over the boundaries
k kth general subscript of w
l lth general subscript of w
m mth reactant (gas or solid)
n nth reaction or nth gaseous reactant
q qth iteration step
s sth component of the networks's output vector
u uth network's neuron in the hidden layer
v vth component of the network's input vector

Superscripts
i ith sampling point into the integration domain
j jth sampling point over the boundaries' integration domain
n number of dimensions of vector z

Greek letters
α stoichiometric coefficient
η adaptative learning rate
ρ density of the solid reactant (g/cm^3)

References

[1] B. Ph van Milligen, V. Tribaldos, J.A. Jiménez, Neural network differential equation and plasma equilibrium solver, Physical Review Letters 75 (1995) 3594–3597.

[2] C. Monterola, C. Saloma, Characterizing the dynamics of constrain physical systems with unsupervised neural network, Physical Review E 57 (1998) 1247R–1250R.

[3] C. Monterola, C. Saloma, Solving the nonlinear Schrodinger equation with an unsupervised neural network, Optics Express 9 (2001) 72–84.

[4] M. Quito, Jr., C. Monterola, C. Saloma, Solving N-body problems with neural networks, Physical Review Letters 86 (2001) 4741–4744.

[5] D.R. Parisi, M.A. Laborde, Modeling steady state heterogeneous gas–solid reactors using feedforward neural networks, Computer and Chemical Engineering 25 (2001) 1241–1250.

[6] D.A. Spretcher, On the structure of continuous function of several variables, Transactions of the American Mathematical Society 115 (1965) 340.

[7] Hetch-Nielsen R. Kolmogorov's mapping neural networks existence theorem, 1st IEEE Inter. Conf. on Neural Networks, San Diego CA, 3 (1987) 11.

[8] R. Hetch-Nielsen, Theory of backpropagation neural network, Proc. Int. Joint Conf. on Neural Networks, Washington D.C. New York, IEEE Press, 1989, pp. I.593–I.605.

[9] G. Cybenko, Approximation by superposition of a sigmoidal function, Mathematics of Control, Signals and Systems 2 (1989) 303.

[10] K.I. Funahashi, On the approximate realization of continuous mappings by neural networks, Neural Networks 2 (1989) 183.

[11] K. Hornik, M. Stichcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Networks 2 (1989) 359.

[12] K. Hornik, M. Stichcombe, H. White, Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks, Neural Networks 3 (1990) 551.

[13] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural Networks 4 (1991) 251.

[14] R. Hetch-Nielsen, Neurocomputing, Addison Wesley Publishing Company, 1990.

[15] J. Hertz, A. Krogh, R.G. Palmer, Introduction to the Theory of Neural Computation, Addison-Wesley Publishing Company, 1991.

[16] J.A. Freeman, D.M. Skapura, Neural Networks: Algorithms, Applications, and Programming Techniques, Addison-Wesley Publishing Company, 1991.

[17] M.T. Hagan, H.B. Demuth, M.H. Beale, Neural Network Design, PWS Publishing, Boston, MA, 1996.

[18] S. Haykin, Neural Networks: A Comprehensive Foundation, 2nd ed., Prentice-Hall Inc, 1999.

[19] J.D. Schaffer, D. Whitley, L.J. Eshelman, Combinations of genetic algorithms and neural networks: a survey of the state of the art, in: L.D. Whitley, J.D. Schaffer (Eds.), COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks, IEEE Computer Society Press, 1992.

[20] D. Whitley, T. Hanson, Optimizing Neural Networks Using Faster, More Accurate Genetic Search, in: J.D. Schaffer (Ed.), Proceedings of the Third International Conference on Genetic Algorithms (Arlington, 1989), San Mateo, Morgan Kauffmann, 1989, pp. 391–396.

[21] D.J. Montana, L. Davis, Training Feedforward Networks Using Genetics Algorithm, in: N.S. Sridharan (Ed.), Eleventh International Joint Conference on Artificial Intelligence (Detroit, 1989), San Mateo, Morgan Kauffmann, 1989, pp. 762–767.

[22] B. Martín del Brio, A. Sanz Molina, Redes Neuronales y Sistemas Borrosos, Ra-Ma, 1997.

[23] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley, 1989.

[24] M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1999.

[25] G.F. Froment, K.B. Bischoff, Chemical Reactor Analysis and Design, John Willey & Sons Inc, 1979.

[26] J.R. Dormand, P.J. Prince, A family of embedded Runge–Kutta formulae, J. Comp. Appl. Math. 6 (1980) 19.