TCAD Modeling Using a Neural Network Based Approach

R. Matei, G. Dima and M.D. Profirescu

EDIL R&D Centre, University Politehnica of Bucharest
PO Box 74-121, 77400 Romania, devsim@edil.pub.ro

ABSTRACT

In this paper we present a neural network based approach for TCAD empirical modeling. We develop a neural interpolator with a multi-layered feed-forward structure for building empirical models based on a set of experimental/simulated data. We assess the neural interpolator performances on modeling the transport parameters for bulk $In_xGa_{1-x}As$.

Keywords: RSM, neural network, empirical modeling

1 INTRODUCTION

Simulation frameworks allow to conduct a planned computer experiment and to build empirical models from a set of experimental/simulated data collected accordingly with a design of experiment (DOE) strategy. The traditional surface modeling [1,2,3] may not be able to fit well on some cases, hence the motivation for searching alternative solutions, which may prove more convenient for particular situations. Neural networks could be such a candidate for the TCAD empirical modeling purposes [4,5].

The present approach tried to combine the precision of a Monte Carlo (MC) algorithm with the flexibility and speed of a neural network structure. The challenge was to find a way to decrease the computational overhead associated with MC simulator while transferring it to the neural network. Classically, the MC simulator compute the values of various output parameters in each simulation point without using the correlation between previous and actual simulating point or in other words without any interference. It would be better to use the MC simulator only in certain points while letting the neural network to learn the nature of the process from them and build an empirical model.

2 MATHEMATICAL MODEL

The neural network is based on a multi-layered perceptron feed-forward architecture trained based on a back-propagation error algorithm. More concrete, we will use an quasi-Newton training sequence optimized for speed, known as Levenberg-Marquardt rule. Basically, this method traces a minimum point on a quadratic cost surface using a classical Newton method but approximating the inverse of the hessian matrix instead of effectively computing it. This leads to a low amount of computing time while preserving precision. The learning process in feed-forward architectures can be studied as an approximation process of a generic function: $F : A \subset R^M \to R^N$ from a bounded region of a M dimensional euclidean space to a N dimensional bounded sub-region $F(A)$ in the same euclidean space. This approximation problem is often solved using supervised learning algorithms and defining a training set: (x_1,d_1), (x_2,d_2) ...(x_P,d_P) where $d_p=F(x_p)$, with $1 \leq p \leq P$. The neural model is also known as "mapping network". The approximation theory has as studying object the approximation of a continuous multivariable function F by the approximation function $F(w,x)$ with a given number of w parameters. For a given function F this w set of parameters must be calculated in order to best fit the F function on a given train set. In our case, the w set represents the weights of the connections between neural cell units. The w set is computed by searching in the entire weights space (this often leads to a NP problem with an exponential solving time because of the high dimension of this space). The solution, noted as w^* (the optimal vector of the network parameters which minimizes a cost function) is not unique due to the possible weights and hidden neural cells permutations therefore there are many points of global minimum and even more of local minimum. There are several precautions to be made in order to prevent algorithm blocking in such points of local minimum: weights symmetry breaking, structure symmetry breaking, carefully choosing the initial conditions [6]. If we identify $w=[w_1,w_2,...w_K]^T$ as the vector of the neural network parameters then we can assume that computing the minimum value of a cost function $C(w)$ (usually a L_2 class function) is a linear, discrete time searching. This searching starts from w_0 following the $\boldsymbol{d}$ direction (the negative gradient of $C(w)$) this gradient being evaluated at every k step by a deterministic first order gradient relation:

$$\boldsymbol{d}_k = -\nabla C(w_k) \tag{1}$$

The w_k parameter is defined as follows:

$$w_{k+1} = w_k + \eta_k \boldsymbol{d}_k \tag{2}$$

where η_k is learning rate at k step. Consecutively steps following the negative gradient are orthogonal resulting in a crisscrossed trajectory:

$$0=\frac{\partial}{\partial\eta_k}C(w_k+\eta_k d_k)=d_k\cdot\nabla C(w_{k+1}) \tag{3}$$

This method is a very fast and simple one but sensible to local minimum problem. The second order deterministic algorithms (as that used by us) involve the second grade derivatives of the cost function $C(w)$

$$d_k=-\nabla C(w_k)+\beta\, d_{k-1} \tag{4}$$

$$d_0=-\nabla C(w_0) \tag{5}$$

where β_k is chosen in order to minimize previous gradient direction variation. This way, the searching path defined by d_k must not alter the gradient component that followed the previous trajectory d_{k-1}. Therefore, till the first derivative depending on η_k, the following relation must be true:

$$d_{k-1}\cdot\nabla C(w_k+\eta_k d_k)=0 \tag{6}$$

The cost function $C(w)$ can be approximated with a Taylor series around the w_k point:

$$\begin{aligned}C(w)&=C(w_k)+(w-w_k)^T\nabla C(w_k)+\\&+\frac{1}{2}(w-w_k)^T\nabla^2 C(w_k)(w-w_k)\end{aligned} \tag{7}$$

We note $J(w_k)$ as the jacobian and $H(w_k)$ as the hessian both evaluated in w_k point:

$$J(w_k)=\nabla C(w_k)=\left.\frac{\partial C}{\partial w_r}\right|_{w_k} \tag{8}$$

$$H(w_k)=\nabla^2 C(w_k)=\left.\frac{\partial^2 C}{\partial w_r\partial w_s}\right|_{w_k} \tag{9}$$

Differentiating (7) and using (8) and (9) results:

$$\nabla C(w_k)=J(w_k)+H(w_k)(w-w_k)+\dots \tag{10}$$

Using (3) and (10) results:

$$d_{k-1}Hd_k=0 \tag{11}$$

Choosing β_k from (4) can be done using various methods (Fletcher-Reeves, Polak-Ribiere). The idea of Newton method on which fundament of quasi-Newton resides is that the minimum of the cost function $C(w)$ which represents the solution of the optimization problem is a point where $\nabla C(w)=0$ and this point can be computed equalizing (10) with 0, in the same time ignoring the higher order terms of the series. If the inverse of the hessian exists then results the discrete time Newton relations:

$$w=w_k-H^{-1}(w_k)J(w_k) \tag{12}$$

$$w_{k+1}=w_k-H^{-1}(w_k)J(w_k) \tag{13}$$

These relations lead to a fast convergence algorithm also with the advantage of avoiding local minimum problem. Due to its disadvantages (hessian might be singular and the method necessitates the computing of the second order derivatives) we tried to approximate the hessian inverse therefore using a quasi-Newton method where $\hat{H}^{-1}$ is the matrix which approximates the hessian inverse $\hat{H}^{-1}$ (k index marks discrete time step k). For w_{k+1} results in:

$$w_{k+1}=w_k+\eta_k d_k=w_k-\eta_k\hat{H}^{-1}\nabla C(w_k) \tag{14}$$

with η_k chosen by minimizing

$$\eta_k=\arg\min_{\eta\geq 0}C\left(w_k-\eta_k\hat{H}^{-1}\nabla C(w_k)\right) \tag{15}$$

and inverse hessian approximation given by relation:

$$\begin{aligned}&\hat{H}^{-1}_{k+1}=\hat{H}^{-1}_k+\\&+\Psi\left(\hat{H}^{-1}_k,w_{k+1}-w_k,\nabla C(w_{k+1})-\nabla C(w_k)\right)\end{aligned} \tag{16}$$

Function Ψ is not a simple one. Therefore can be seen that no evaluation of the actual hessian inverse is performed. This quasi-Newton method in searching path regarding gradient descent rule is stable and the amount of memory required is lower than that required by the classical Newton, all of these while keeping a high speed. An observation regarding the local minimum should be made: this problem isn't so relevant because almost always the evaluation of $C(w)$ is made with an imposed error. When imposing the error we do not know which is the relation between this error and the global minimum error, which might exists or not in a given case (the complexity of the error surface in a multidimensional space does not allow assertion like "there is a global minimum" or "there is only a global minimum" to be considered reasonable). Therefore, for a given problem the global minimum is the imposed error and often it is not interesting if somewhere on the error surface exists another point where the error is much smaller than that imposed as long as my performance request was satisfied [6].

3 SIMULATION FRAMEWORK

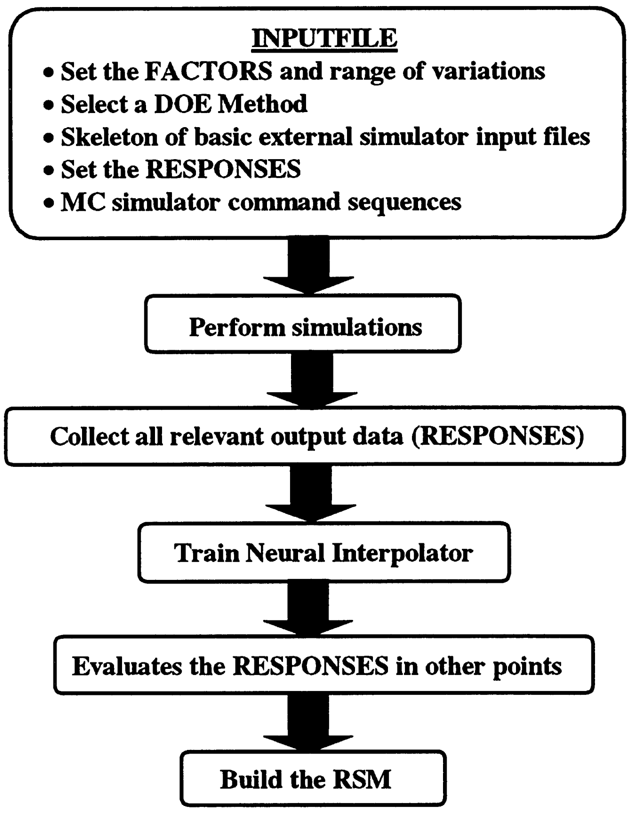
INPUTFILE
• Set the FACTORS and range of variations
• Select a DOE Method
• Skeleton of basic external simulator input files
• Set the RESPONSES
• MC simulator command sequences
Perform simulations
Collect all relevant output data (RESPONSES)
Train Neural Interpolator
Evaluates the RESPONSES in other points
Build the RSM

Figure 1: The used simulation framework

Of course some aspects should be commented. First, this approach is efficient only if the total number (*N*) of points that are to be simulated is very high. Then, it make sense to select from them only the relevant ones or in any case a number *M* significantly smaller than *N*. As the selection criteria can be used a DOE strategy linked with a classic PCA (Principal Component Analysis) or NPCA (Nonlinear PCA) algorithm, the last above very well suited especially in multidimensional non-linear spaces. The problems often raised in training phase are avoiding blocking in a local minimum, avoiding slow convergence speed or non-convergence and maintaining a low overhead on computing resources (memory, CPU time etc.) [6]. Solutions to this problems were mentioned also in the second chapter and many of them are fully functional in today simulations tools with very good results.

As developing platform for the neural interpolator (NI) we used MatLab 5.1 with Neural Networks Toolbox and as Monte-Carlo solver a bulk Monte Carlo simulator.

The neural network was build from three layers (one hidden layer) with *L1-L2-L3* (*L1* factors, *L3* targets and *L2* hidden neural cells). This neural structure was trained using a Levenberg-Marquardt algorithm with a MSE (Mean Squared Error) performance function *C(w)*. Neural cell transfer function used was a biased sigmoid.

4 RESULTS

We applied this approach to bulk $Ga_{1-x}In_xAs$ analyzing/extracting three output parameters (responses) as functions of three variables (factors): the mole fraction (x), the doping concentration (N_D) and the intensity of electric field. The set of responses consisted of the average electron velocity in the direction of electric field (AEV), average electron energy (AEE), effective electron mass (EEM). The train set was obtained by running a bulk Monte Carlo simulator for 27 sets of factors given by a 3 level full factorial DOE.

Following consequently the purpose of increasing speed we choose to build separate (architectural identically) network structures for each parameters therefore the correlation existing between output parameters was neglected. This approach is somehow similar with the clustering technique used in some neural architectures (this prevents a mutual negative influence of various output parameters with different ranges, eliminates undesired connection redundancy and speeds up training process). It is worth to mention that even in these circumstances the memory amount required for keeping a number of seven or eight network structures is not relevant.

In Figure 2 is shown the AEV versus electric field for an arbitrarily set of input values (x=0.275 and N_D=1.e15cm^{-3}) not considered in the train set. The training time was 2.86 seconds, test time was 0.05 seconds (train error 0.005, test error 0.0036). The average speed-up calculated for 10 extractions was 7.81. Similar performance was encountered for the AEE (Figure 3) and EEM (Figure 4).

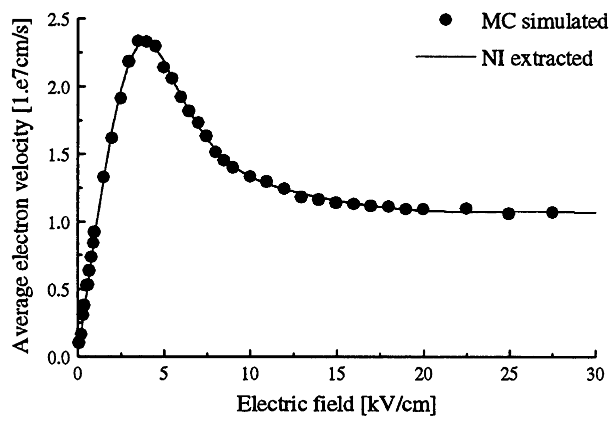

Figure 2: The average electron velocity versus electric field for x=0.275 and N_D=1e15 cm^{-3}

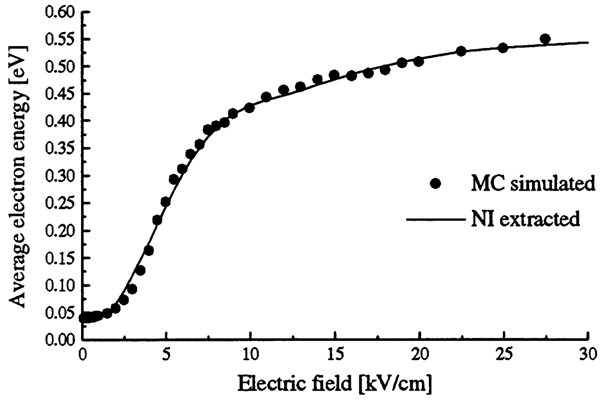

Figure 3: The average electron energy versus electric field for x=0.275 and N_D=1e15 cm^{-3}

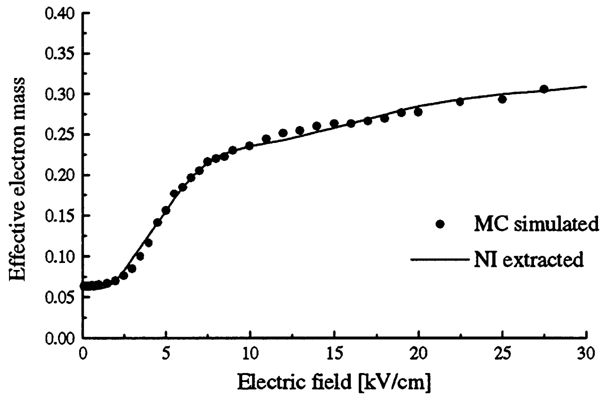

Figure 4: The effective electron mass versus electric field for x=0.275 and N_D=1e15 cm^{-3}

In Figure 5 and 6 are shown the AEV and AEE empirical models build by the neural interpolator versus mole fraction and electric field. The response time in the case of a single point was practically 0.05 seconds with a test error of 0.01.

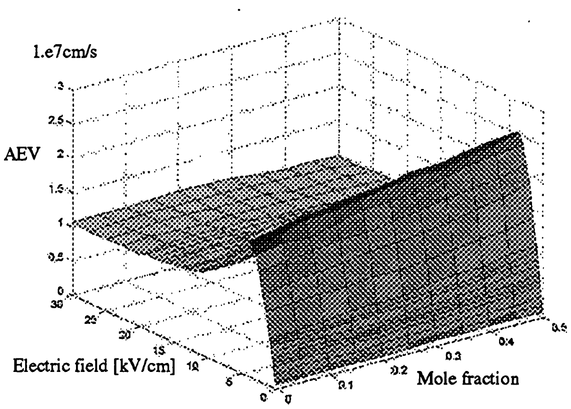

Figure 5: The average electron velocity versus electric field and mole fraction for N_D=1e15 cm^{-3}

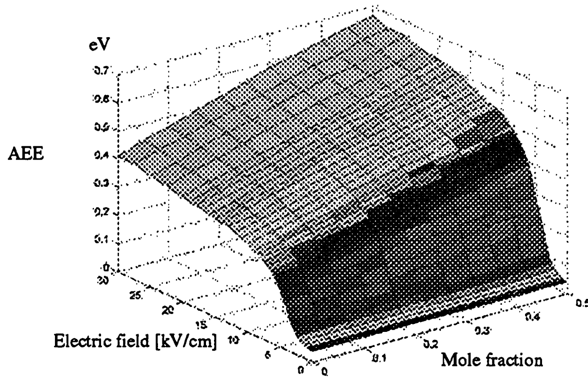

Figure 6: The average electron energy versus electric field and mole fraction for N_D=1e15 cm^{-3}

5 CONCLUSIONS

We developed a technique for building RSM using a NI based on a multi-layered perceptron feed-forward architecture trained with a quasi-Newton algorithm. The training data set was selected using DOE principles. This technique was assessed on modeling the average electron velocity, energy and effective mass for bulk $In_xGa_{1-x}As$ as functions of mole fraction, electric field and doping concentration.

By coupling a bulk MC simulator and the NI in a simulation framework we obtained an efficient speed-up methodology.

ACKNOWLEDGEMENTS

The Romanian Academy is kindly acknowledged for the financial support of this work.

REFERENCES

[1] R. Myers, A.I. Khuri, and Jr. W.H. Carter, "Response Surface Methodology", Technometrics, 31(2):137-157, 1989.

[2] W. Schoenmaker and R. Cartuyvels, "Theory and Implementation of a New Interpolation Method Based on Random Sampling", IEEE J. TCAD. URL: http://tcad.stanford.edu/tcad-journal/archive/, 1997.

[3] B. Govoreanu, W. Schoenmaker, G. Kopalidis, O. Mitrea, G. Dima, and M.D. Profirescu, "A Hybrid Technique for TCAD Modeling&Optimization", IEEE J. TCAD. URL: http://tcad.stanford.edu/tcad-journal/archive/, 2000.

[4] D.J.C. MacKay, "Bayesian Interpolation", Neural Computation, 4(3):415-447, 1992.

[5] B. Govoreanu, J. Suykens, W. Schoenmaker, C. Amza, G. Dima, J. Vandewalle, and M.D. Profirescu, "A Comparison Between Various Empirical Models for TCAD Purposes" Proceedings of CAS2000, pp. 315-318, 2000.

[6] J.A. Freeemam and D. M. Skapura, "Neural Networks–Algorithms, Applications and Programming Technique", Addison-Wesley Publishing Company, New York, 1991.