Neural Networks for Device and Circuit Modelling

Peter B.L. Meijer

Philips Research Laboratories, Eindhoven, The Netherlands
E-mail: Peter.B.L.Meijer@philips.com

Abstract. The standard backpropagation theory for static feedforward neural networks can be generalized to include continuous dynamic effects like delays and phase shifts. The resulting non-quasistatic feedforward neural models can represent a wide class of nonlinear and dynamic systems, including arbitrary nonlinear static systems and arbitrary quasi-static systems as well as arbitrary lumped linear dynamic systems. When feedback connections are allowed, this extends to arbitrary nonlinear dynamic systems corresponding to equations of the general form $\boldsymbol{f}(\boldsymbol{x}, \dot{\boldsymbol{x}}, t) = \mathbf{0}$. Extensions of learning algorithms to include combinations of time domain and frequency domain optimization lead to a semi-automatic modelling path from behaviour to simulation models. Model generators have been implemented for a range of existing analog circuit simulators, including support for the VHDL-AMS and Verilog-AMS language standards.

1 Introduction

With the continuing advances in digital technology, now with deep-submicron devices and wires switching at such high frequencies that RF effects appear, it becomes increasingly hard to maintain a clear separation between the digital abstraction from 1's and 0's upward and the underlying physical world which is characterized by analog behaviours. Together with the rapidly increasing complexity of digital and mixed-signal designs, this calls for methodologies that further support reuse as well as the abstraction from analog behaviour at the device level to intermediate levels of abstraction. Our aim is to find simplified simulation models while sufficiently preserving the analog functional behaviour in order to allow for further testing and validation of designs and design rules through simulation. New language standards like VHDL-AMS and Verilog-AMS help to co-simulate analog models in a digital environment, but the task of finding efficient and sufficiently accurate analog and mixed-signal models often remains a daunting one.

The methodology as outlined in this paper may on the one hand assist in quickly obtaining accurate analog simulation models where no suitable or efficient physical device models are available (yet), while on the other hand it may find applications in macro-modelling for mixed-level mixed-signal simulation where structural models already do exist in the form of netlists of interconnected transistors and other devices, but not or not yet in the form of more efficient simplified functional models.

The proposed approach is based on a generalization of feedforward neural networks, also known as multilayer perceptron (MLP) networks [6]. We add time differentiation to the connecting weights and use nonlinear transfer functions and time integration in the neuron bodies to arrive at a modelling formalism capable of representing the time-dependent nonlinear behaviour of a very wide class of electronic circuits. Learning of dynamic nonlinear multivariate behaviour can be done with combinations of time domain and frequency domain data. However, since the approach is still ultimately rooted in continuous optimization algorithms for minimizing errors between supplied behavioural data and model, it does not on its own take away the intrinsic problems associated with all known continuous nonlinear optimization methods, such as a possibly slow convergence (when far from the optimum point) and the risk of being trapped in some local minimum. These problems, if they occur, can be circumvented or alleviated by using suitable templates for initializing the neural network topology and parameters. This hybrid approach will be outlined and illustrated in sections 3 and 4.

2 Neural Network Equations

The neural networks used in this paper are defined in this section. A detailed motivation for the various specific choices can be found in [4]. Layers are counted starting with the input layer as layer 0, such that a network with output layer K involves a total of $K+1$ layers. Layer k by definition contains N_k neurons. A vector notation with bold font is used to denote information on all neurons in a particular layer. A neural network has a vector of inputs $\boldsymbol{x}^{(0)}$ and a vector of outputs $\boldsymbol{x}^{(K)}$.

The differential equation for the output, or excitation, y_{ik} of one particular neuron i in layer $k > 0$ is given by

$$\tau_{2,ik} \, \frac{\mathrm{d}^2 y_{ik}}{\mathrm{d}t^2} \; + \; \tau_{1,ik} \, \frac{\mathrm{d}y_{ik}}{\mathrm{d}t} \; + \; y_{ik} \; = \; \mathcal{F}^{(ik)}(s_{ik}, \delta_{ik}) \tag{1}$$

with timing parameters $\tau_{1,ik}$ and $\tau_{2,ik}$, and the source term $\mathcal{F}^{(ik)}$ a (generally nonlinear) function having an optional transition parameter δ_{ik}. The weighted sum s_{ik} of results from the preceding layer is further defined as

$$\begin{aligned} s_{ik} &\stackrel{\triangle}{=} \boldsymbol{w}_{ik} \cdot \boldsymbol{y}_{k-1} - \theta_{ik} \; + \; \boldsymbol{v}_{ik} \cdot \frac{\mathrm{d}\boldsymbol{y}_{k-1}}{\mathrm{d}t} \\ &= \sum_{j=1}^{N_{k-1}} w_{ijk} \, y_{j,k-1} \; - \; \theta_{ik} \; + \; \sum_{j=1}^{N_{k-1}} v_{ijk} \, \frac{\mathrm{d}y_{j,k-1}}{\mathrm{d}t} \end{aligned} \tag{2}$$

for $k > 1$, involving weighting parameters w_{ijk} and v_{ijk}, an offset parameter θ_{ik}, and similarly for the neuron layer $k = 1$ connected to the network inputs

$x_j^{(0)}$

$$\begin{aligned} s_{ik} &\stackrel{\triangle}{=} \boldsymbol{w}_{ik} \cdot \boldsymbol{x}^{(0)} - \theta_{ik} \; + \; \boldsymbol{v}_{ik} \cdot \frac{\mathrm{d}\boldsymbol{x}^{(0)}}{\mathrm{d}t} \\ &= \sum_{j=1}^{N_0} w_{ij,0}\, x_j^{(0)} \; - \; \theta_{i,0} \; + \; \sum_{j=1}^{N_0} v_{ij,0} \, \frac{\mathrm{d}x_j^{(0)}}{\mathrm{d}t} \end{aligned} \tag{3}$$

which is analogous to having a dummy neuron layer $k = 0$ with enforced neuron j outputs $y_{j,0} \equiv x_j^{(0)}$, or in vector notation $\boldsymbol{y}_0 \equiv \boldsymbol{x}^{(0)}$.

Finally, to allow for arbitrary network output ranges in case of bounded functions $\mathcal{F}^{(ik)}$, a linear scaling transformation is added to the output stage

$$x_i^{(K)} \; = \; \alpha_i \, y_{iK} \; + \; \beta_i \tag{4}$$

yielding a network output vector $\boldsymbol{x}^{(K)}$.

The above function $\mathcal{F}^{(ik)}$ is for neuron i in layer k applied to the weighted sum s_{ik} of neuron outputs $y_{j,k-1}$ in the preceding layer $k-1$. The optional transition parameter δ_{ik} may be used to set an appropriate scale of change in qualitative transitions in function behaviour, as is common to semiconductor device modelling. For example, choosing the following function $\mathcal{F}_2$ as

$$\mathcal{F}^{(ik)}(s_{ik}, \delta_{ik}) \; = \; \mathcal{F}_2(s_{ik}, \delta_{ik}) \stackrel{\triangle}{=} \frac{1}{\delta_{ik}^2} \; \ln \frac{\cosh \dfrac{\delta_{ik}^2 (s_{ik} + 1)}{2}}{\cosh \dfrac{\delta_{ik}^2 (s_{ik} - 1)}{2}} \tag{5}$$

lets the parameter δ_{ik} be used to optimize the transition between a nearly linear region and two asymptotically exponential tails. The function $\mathcal{F}_2$ is illustrated in Fig. 1.

Again referring to [4], it can be shown that the above neural network equations can represent any lumped linear dynamic system, and can arbitrarily closely approximate any multivariate static (DC) or quasistatic model—such as typically used to model the DC and capacitive currents of MOSFET transistors. Under very weak conditions, this approximation property applies even when the $\mathcal{F}^{(ik)}$ are the same for all neurons, while requiring only three layers for static models [1–3] and at most four layers for quasistatic models.

Moreover, when allowing for external feedback connections from the outputs of the dynamic feedforward neural network as defined above back to

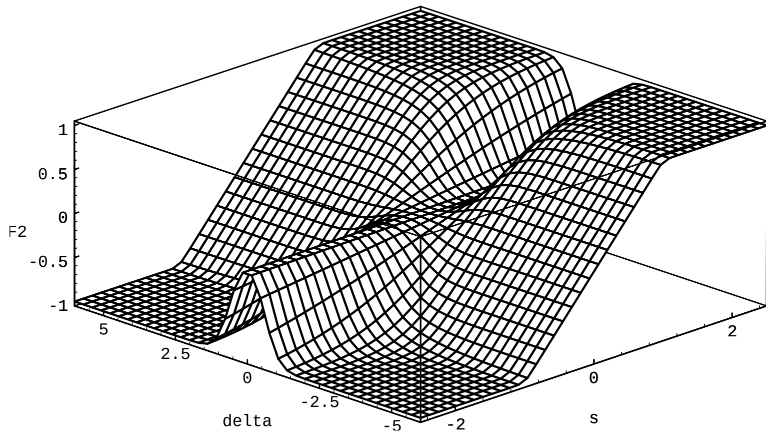

Fig. 1. Neuron nonlinearity $\mathcal{F}_2(s_{ik}, \delta_{ik})$.

the network inputs, it can be shown[1] that the resulting class of neural networks can represent arbitrary nonlinear dynamic systems corresponding to equations of the general form $\boldsymbol{f}(\boldsymbol{x}, \dot{\boldsymbol{x}}, t) = \mathbf{0}$, thus covering a very broad class of multidimensional nonlinear non-quasistatic systems [4,5]. This includes systems having multiple DC solutions, which the above-defined feedforward neural networks cannot represent without these additional feedback connections.

3 Initialization Templates and Learning Phase

Apart from using one and the same nonlinearity for all neurons, one may also select an appropriate function $\mathcal{F}^{(ik)}$ for each neuron individually, for instance by using initializing templates that capture any available a priori knowledge about the approximate behaviour or structure of the device or circuit to be modelled. Furthermore, once known, the set of poles and zeros of any linear transfer function can be mapped exactly and constructively to corresponding linear(ized) neural (sub)networks [4]. Univariate transcendental functions, such as sine, square root and exponential, are readily associated

[1] The existence proof builds on the theorems for static models [1–4]. By defining a companion function $\boldsymbol{F}(\boldsymbol{x}, \dot{\boldsymbol{x}}, t)$ we may arbitrarily closely approximate the *static* (w.r.t. the direct arguments $\boldsymbol{x}$, $\dot{\boldsymbol{x}}$ and t) function expression $\boldsymbol{f}(\boldsymbol{x}, \dot{\boldsymbol{x}}, t) + \boldsymbol{x}$, by making use of the time differentiation available in the input connections to the neural network and adding time as an additional input, the latter either directly or via a time dependent vector function $\boldsymbol{u} = \boldsymbol{u}(t)$. Next closing the feedback connections for $\boldsymbol{x}$ to obtain $\boldsymbol{f}(\boldsymbol{x}, \dot{\boldsymbol{x}}, t) + \boldsymbol{x} \equiv \boldsymbol{x}$ completes the implicit system equivalent to the state equation $\boldsymbol{f}(\boldsymbol{x}, \dot{\boldsymbol{x}}, t) = \mathbf{0}$ that was to be modelled. Any separate output equations of the general form $\boldsymbol{y} = \boldsymbol{G}(\boldsymbol{x}, \boldsymbol{u}, \dot{\boldsymbol{u}})$, can similarly be accounted for and included in the neural network.

with corresponding instances of $\mathcal{F}^{(ik)}$, while the (bivariate) multiplication of two sub-expressions is easily mapped to a neural (sub)network through a linear combination of scalar squaring functions as in $xy = \frac{1}{4}[(x+y)^2 - (x-y)^2]$, using two instances of $\mathcal{F}^{(ik)}$ for the squaring operation. One can apply these basic mappings to create templates manually, but one can also devise a general parser for automatically mapping differential-algebraic systems of equations onto exactly equivalent dynamic neural networks.

After the optional initialization phase, where topologies and parameters are set through a template containing a priori modelling knowledge, the actual optimization ("learning") starts. This involves a continuous optimization algorithm for minimizing a cost function for the model error w.r.t. the supplied behavioural data. This data in turn can be a mixture of DC, time domain and frequency domain data. The latter may include scatter parameters obtained from small signal AC analysis or measurements. Many optimization algorithms, such as conjugate gradient and BFGS, need the gradient of the cost function, and this implies the use of DC sensitivity, transient sensitivity and AC sensitivity. A basic time integration algorithm such as Backward Euler can be used for the discretized transient analysis and transient sensitivity. Higher order algorithms may be applied for greater simulation efficiency at the expense of increased algorithmic complexity.

4 Using A Priori Knowledge—An Example

To illustrate the use of a priori knowledge in macro-modelling a complex circuit by means of dynamic neural networks, we will consider the folding part of a folding AD converter designed within Philips. This circuit block has one input and 32 outputs. Within the operating range of interest, the DC output of each of these outputs is approximately sinusoidal as a function of the input. Different outputs differ only by subsequent phase steps of $\pi/32$. Therefore, we can make use of $\sin(x + \phi_i) = c_{1i} sin(x) + c_{2i} sin(\pi/2 - x)$ with $c_{1i} = \cos(\phi_i)$, $c_{2i} = \sin(\phi_i)$, to write all the DC outputs as linear combinations of just two instances of sine functions. Next, the neural network of choice was correspondingly initialized as a 1-2-32 network, with two instances of $\mathcal{F}^{(i1)}(s_{i1}, \delta_{i1}) = \sin(2\pi s_{i1})$ in the single "hidden" layer $k = 1$, and the identity functions $\mathcal{F}^{(i2)}(s_{i2}, \delta_{i2}) = s_{i2}$ for the 32 outputs in layer $k = 2$. The neural network topology is illustrated in Fig. 2.

The initial values for $\tau_{1,ik}$ were initially all identical and estimated from graphical inspection of the high-frequency roll-off of the transistor-level circuit behaviour. The initial values for $\tau_{2,ik}$ were based on those for $\tau_{1,ik}$ while ensuring a small initial quality factor $Q \ll 1$ for each neuron. The details of setting proper values for the weights and offsets will be omitted here. Through subsequent optimization with our neural modelling software, the timing parameters were readily adapted for a much better fit to the actual circuit response curves. The final result of neural network training is illustrated in

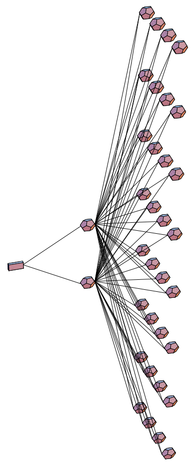

Fig. 2. Neural network topology for 1-2-32 network.

Fig. 3 where model behaviour and circuit behaviour (”Target output”) are shown for one of the 32 outputs (all outputs gave comparable results), and using a sinusoidal time-dependent input signal—which had also been used in the training phase[2]. The higher frequencies in the output signal are the result of the input signal amplitude spanning several periods of the approximately sinusoidal DC response. The model accurately represents the high-frequency roll-off, which causes the non-constant envelope in the output signal, with smaller excursions at the higher frequencies.

The original circuit block contained about five hundred transistors, normally simulated using advanced analog CAD device models, while the neural model now only needs two sine functions and 32 simple linear combinations of these, plus low-order linear differential equations to account for the dynamic behaviour. The computational complexity was therefore lowered by several orders of magnitude, and depending on the simulator used, simulation speedups of two or three orders of magnitude were obtained for simulating the model versus the circuit block. The differential-algebraic equations and parameters of any neural model can be automatically mapped to the syntax supported by the preferred analog circuit simulator. For example, the neuron body as specified in Eq. (1) can be rewritten and translated into corresponding VHDL-AMS code like

[2] In general one should run a cross-validation with data that was not used in the training in order to reduce the risk of accidental over-fitting.

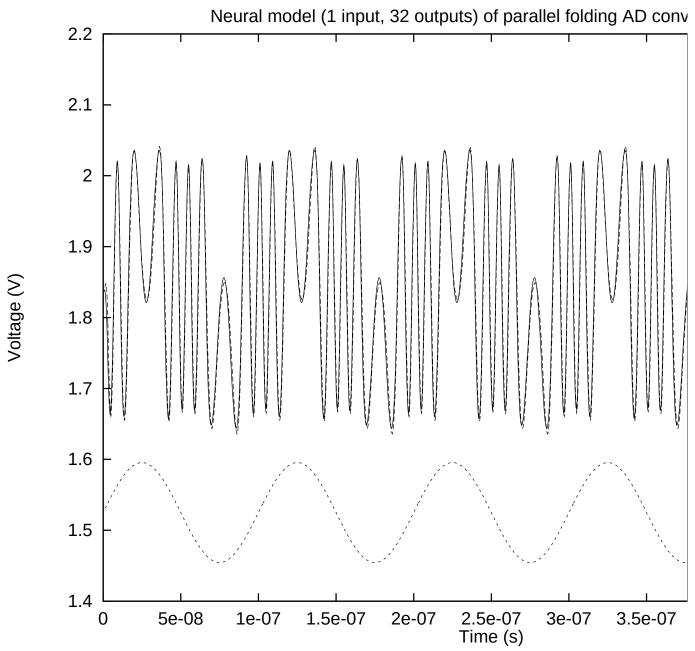

Fig. 3. Simulation with original circuit and neural model.

```
PACKAGE BODY soma_pkg IS

   USE IEEE.MATH_REAL.all;

   -- DC nonlinearity selection and function evaluation
   FUNCTION somadc(ftype: INTEGER; s, delta: REAL) RETURN REAL IS
      VARIABLE f: REAL;
   BEGIN
     CASE ftype IS
     WHEN 0 => f := s;
     WHEN 5 => f := sin(math_2_pi*s);
     WHEN OTHERS => RETURN 0.0;
     END CASE;
     RETURN f;
   END somadc
END soma_pkg;

ENTITY dynsoma IS
   GENERIC (ftype: INTEGER; delta, tau1, tau2: REAL);
   PORT (TERMINAL INA, OUTA, REF: ELECTRICAL);
END ENTITY dynsoma;

ARCHITECTURE soma OF dynsoma IS
   TERMINAL AUX: ELECTRICAL;
   QUANTITY s ACROSS INA TO REF;               -- Weighted sum s
   QUANTITY x ACROSS AUX TO REF;               -- Auxiliary variable
   QUANTITY y ACROSS iout THROUGH OUTA TO REF; -- Neuron output y
BEGIN
   x == tau1 * y'dot;
   y == somadc(ftype,s,delta) - x - tau2/tau1 * x'dot;
END ARCHITECTURE soma;
```

where the function type ftype was used to specify what function $\mathcal{F}$ should be used in the folding AD converter model. A different description is used for model instances with zero-valued τ_1. Similarly, model code can be generated for Verilog-AMS or any other sufficiently rich simulation or programming language. Consistency among models in different simulation languages is always ensured through the automatic mapping by model generators, which all use the same network topology and parameter set.

5 Conclusions

Dynamic neural networks can be used for a wide range of device and circuit modelling applications. The generalized formalism allows for a hybrid modelling approach where existing knowledge can be incorporated before starting the general optimization or learning phase. This approach can help to trade off the respective strengths and weaknesses of physical modelling and automatic black-box behavioural modelling.

References

1. K.-I. Funahashi, "On the Approximate Realization of Continuous Mappings by Neural Networks," *Neural Networks*, Vol. 2, pp. 183-192, 1989.
2. K. Hornik, M. Stinchcombe and H. White, "Multilayer Feedforward Networks are Universal Approximators," *Neural Networks*, Vol. 2, pp. 359-366, 1989.
3. M. Leshno, V. Y. Lin, A. Pinkus and S. Schocken, "Multilayer Feedforward Networks With a Nonpolynomial Activation Function Can Approximate Any Function," *Neural Networks*, Vol. 6, pp. 861-867, 1993.
4. P. B. L. Meijer, "Neural Network Applications in Device and Circuit Modelling for Circuit Simulation," *Ph.D. thesis*, Eindhoven University of Technology, May 2, 1996.
5. P. B. L. Meijer, "Signal generator for modelling dynamical system behaviour,", *U.S. patent*, No. 5790757, August 4, 1998.
6. D. E. Rumelhart and J. L. McClelland, Eds., *Parallel Distributed Processing, Explorations in the Microstructure of Cognition.* Vols. 1 and 2. Cambridge, MA: MIT Press, 1986.