Article

Deep Learning Neural Network Algorithm for Computation of SPICE Transient Simulation of Nonlinear Time Dependent Circuits

David Černý and Josef Dobeš *

Department of Radio Electronics, Czech Technical University in Prague, Technická 2,
166 27 Praha, Czech Republic; cernyd1@fel.cvut.cz

* Correspondence: dobes@fel.cvut.cz

Abstract: In this paper, a special method based on the neural network is presented, which is conveniently used to precompute the steps of numerical integration. This method approximates the behaviour of the numerical integrator with respect to the local truncation error. In other words, it allows the precomputation of the individual steps in such a way that they do not need to be estimated by an algorithm but can be directly estimated by a neural network. Experimental tests were performed on a series of electrical circuits with different component parameters. The method was tested for two integration methods implemented in the simulation program SPICE (Trapez and Gear). For each type of circuit, a custom network was trained. Experimental simulations showed that for well-defined problems with a sufficiently trained network, the method allows in most cases reducing the total number of iteration steps performed by the algorithm during the simulation computation. Applications of this method, drawbacks, and possible further optimizations are also discussed.

Keywords: SPICE; numerical integration; neural network; step estimation; tensorflow

Citation: Černý, D.; Dobeš, J. Deep Learning Neural Network Algorithm for Computation of SPICE Transient Simulation of Nonlinear Time Dependent Circuits. *Electronics* **2022**, *11*, 15. <https://doi.org/10.3390/electronics11010015>

Academic Editor: Maciej Ławryńczuk

Received: 29 November 2021

Accepted: 20 December 2021

Published: 22 December 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

There is currently a huge expansion in the development of artificial intelligence (AI), neural network (NN), and machine learning (ML). It is amazing how far the field has come in the last decade. Nowadays, we encounter algorithms based on NN in everyday devices such as cars, cell phones, or smart home systems. They have become an integral part of our lives. They help us with navigation, speech recognition, coloring black and white pictures, solving hard optimization problems, and more [1,2]. NN and AI algorithms slowly make their way into disciplines long believed to be the unique domain of humans, such as literature and art [3]. A whole new segment of the industry has emerged, offering solutions that analyze, learn, and generalize work procedures of company employees, and then take over the activities that AI would be capable of solving on its own [4]. In this paper, we present a method for improving the computation performance of Transient Analysis of Simulation Program with Integrated Circuit Emphasis (SPICE) simulators [5] based on Numerical Integration (NI) step estimation with the utilization of NN.

Numerical integration itself has many aspects that can be improved and changed in some way. One can find papers dealing with the complete replacement of numerical integration by another method, in this case, based on the neural network [6]. By modifying the calculation of the predictor and the corrector [7,8], papers publishing different methods for determining the integration step [9,10], and a paper modifying the calculation method using logarithmic damping [11], all of these papers, as well as this paper, share the common goal of speeding up or refining the calculation of numerical integration.

Although many papers have been published on this topic, few methods are suitable for the simulation of electrical circuits. For example, the method presented in the paper [6] would not be suitable and this is due to several aspects. The first one is the low accuracy of the calculation and the second is the speed which is key to the simulation. In this paper, we

focused on modifying the integration step mainly also because the error of computation can be easily determined and fixed. That is, an incorrect determination could at most affect the speed of the computation, but no longer its stability and accuracy. These are key properties of the method provided in this article. A very interesting solution to the integration step calculation can be found in the article [9]. The authors managed to build a very fast algorithm that achieves very good results even for very poorly simulated ones, which switching networks certainly belong to. The problem, however, works well for the low orders of the NI method. The method loses accuracy on smooth parts which in turn affects the increase in the number of integration steps. The paper [10] presents a method for calculating the integration step specifically designed for circuits where the stimulus value is turned off for some time. The algorithm even achieves up to twice the convergence of the standard SPICE program. In our work, we have not been able to achieve such an improvement, but our method proved to be more versatile. The ultimate goal of our proposed method is to increase speed of computation by lowering number of iteration step that diverged altogether with preserved accuracy of the original solution.

The proposed solution was developed exclusively for use in SPICE-like electrical circuit simulation. It has been experimentally tested by implementing it in the circuit simulation program NgSpice [12], which is a freely available open-source alternative to SPICE simulator. It offers a variety of simulation types. From basic Direct Current (DC), Operation Point (OP), Sensitivity Analysis through Monte Carlo, and Noise Analysis [13]. The main motivation to use NgSpice was that its source code is available and thus we could modify them for our purposes.

As mentioned above, in this work we mainly focused on Transient Analysis (TRANS), which simulates the behavior of a circuit over time. In the computational core of the SPICE program the implemented mathematical algorithms are shared among a variety of simulations. It is thus possible to reuse our proposed method for other types of simulations with only a slight modification.

2. SPICE Simulation Process

Before each simulation, SPICE performs the so-called Modified Nodal Formulation (MNF) [14]. This is the process of converting the circuit definition and its simulation into a set of equations. These equations can form a set of linear, non-linear, or even time-dependent equations. Typically, the product of this formulation is a system of nonlinear differential equations of the form

$$F(\mathbf{x}, \dot{\mathbf{x}}, t) = 0, \quad (1)$$

where $\mathbf{x}$ is the vector of unknown circuit variables, $\dot{\mathbf{x}}$ is the derivative of vector $\mathbf{x}$, with respect to time t . In real implementation the values of $\mathbf{x}$ and $\dot{\mathbf{x}}$ depend on current state of the simulation. The first step of the simulation is usually direct current (DC) analysis, that computes operating point of a non-linear circuit. It must occur first to get the linear characteristics of the nonlinear models. However, this step is not always necessary and in these cases, the DC calculation is usually skipped. Very rarely are all devices in a circuit linear, therefore the simulator performs the linearization in the next step using the iterative Newton-Raphson (NR) method and possibly implicit NI. In our paper, we focus specifically on the case where the time-dependent variables need to be removed from the circuit using NI before the algorithm can try to linearize the system using an iterative method. Thus, it must be prepared for calculation by the factorization method. The generalized equation for value $x_n \approx x(t_n)$ where $t_n = t_0 + nh$, with h denoting the step size, can be written as

$$x_{n+1} = \sum_{i=0}^n a_i x_{n-1} + \sum_{i=-1}^n b_i \dot{x}_{n-i}, \quad (2)$$

where coefficients a and b determine the type of integration method and whether the method is explicit or implicit. This step transforms the nonlinear differential equations for each point into a series of non-invariant linear equations. From the problem description it

must already be clear that an error in the linear integration will affect the computation of all subsequent steps, and thus a possible restart of a failed step will negatively affect the overall computational performance of the simulation. Two NI methods are present in the basic version of SPICE. The first NI method is Trapezoidal (TRAPEZ) [15].

The Trapezoidal integration is a second-order method that can be derived based on the observation that a more accurate solution can be obtained if the average of the slopes t_n and t_{n+1} are used as compared to either one or the other:

$$x_{n+1} = x_n + \frac{2}{h}(\dot{x}_{n+1} + \dot{x}_n). \quad (3)$$

The second method available in SPICE is the so-called Backward Differentiation Formula (BDF). In SPICE it is referred to as the GEAR method [16]. All the stable steps of the GEAR method can be derived from the equation

$$\sum_{k=0}^s a_k x_{n+k} = h\beta f(t_{n+s}, x_{n+s}), \quad (4)$$

where a and β are coefficients of the multistep BDF method defined up to order $s < 7$.

As can be seen from Algorithm 1, there is a relatively high chance that a given step will not be accepted, in which case the estimate will have to be rejected and the entire simulation step recalculated with a finer or coarser step. It is important to note that while the above equations work with variables for simplicity, in a real situation, the computations are accompanied by a number of rather demanding computational operations over sparse matrices. Such as pivoting, reordering, and LUF transformations [13]. Incorrect estimates of the integration step significantly slow down the overall computation time of the simulation.

Algorithm 1 Setting New Integration Step

```

Compute  $h_{n+1} = f(LTE)$ 
if ( $h_{n+1} < 0.9 \cdot h_n$ ) then
    reject  $t_{n+1}$ 
    recompute for  $h_n = h_{n+1}$ 
else
    accept  $t_{n+1}$ 
     $h_{n+2} = \min(h_{n+1}, 2h_n, T_{max})$ 
end if

```

3. Neural Network Step Estimator Algorithm

The main goal of the proposed algorithm is to replace the function of the integration step estimation with the Neural Network Step Estimator (NNSE) trained for the calculation of the integration step of specific circuit during a transient simulation. The goal is to decrease the number of iterations by introducing an adaptive step size to the simulation problem. It should be noted that for the final implementation in program SPICE, an extra algorithm for circuit classification would be needed that selects the correct trained NN based on the circuit topology and simulation type. The classification function is not part of the text and it was determined manually. The NN was trained for specific circuits, and then only the circuit settings varied. Therefore, it was not necessary to adequately classify the circuits. Training a general-purpose algorithm that would be applicable to any kind of simulation is up for wider discussion.

The standard SPICE algorithm calculates the step size to achieve the target LTE by the following equation:

$$h_{n+1} = \sqrt{\frac{trtol \cdot \epsilon}{\max\left(\frac{|DD_3(x)|}{12}, abstol\right)}} \quad (5)$$

from which the following step is determined as

$$t_{n+1} = t_n + h_{n+1} \quad (6)$$

where RELTOL is relative error tolerance allowed (Default = 0.001), ABSTOL is absolute error tolerance allowed (Default = 10^{-12}) and TRTOL is transient error tolerance (Default = 7). $\frac{DD_3}{12}$ is divided difference, ϵ a maximum value of heuristically scaled solutions of x and $\dot{x}$. The recursive formula for divided differences is

$$DD_k = \frac{DD_{k-1}(t_{n+1}) - DD_{k-1}(t_n)}{\sum_{i=1}^k h_{n+1-i}} \quad (7)$$

where DD_1 is the numerical approximation of the derivative of x between t_n and t_{n+1}

$$DD_1 = \frac{x_{n+1} - x_n}{h_n} \quad (8)$$

The algorithm evaluates how the chosen integration step by NNSE reflects the rate of change of the simulation. If the step were too small, it would increase the computation time. Conversely, for a too large step, the calculation would lose accuracy.

To replace Equation (5) we chose standard Multilayer Perceptron (MPL) [17] definition of NN that is schematically given in Figure 1. The neurons of the model are connected through their weights. The input layer information is passed through the hidden layers and then the output vector is computed from them. Each neuron performs a weighted summation of its inputs which are basically the outputs of the neurons on the previous layers. This can be expressed by the equation

$$v_m = \sum_{i=1}^L w_{im} x_i + b_i \quad (9)$$

where w_{im} is weight and b_i is threshold value. The input signals are cumulative and the neural block is activated by the the activation function, and has only one output y_m :

$$y_m = \sigma(v_m) \quad (10)$$

Figure 1. Example of the Multilayer Perceptron Network.

The ReLU activation function [18] proved to be the best for our needs. Its only limitation is its discontinuity around 0. The function can be expressed by the following algorithm

$$R(z) = \max(0, z) \quad (11)$$

therefore, the calculation of the NI step has changed to the equation

$$\hat{h}_{n+1} \approx y_m \quad (12)$$

and

$$t_{n+1} = t_n + \hat{h}_{n+1} \quad (13)$$

where $\hat{h}$ characterizes the estimated value of the next NI step by NNSE. The implementation does not affect the basic settings of the method coefficients. Thus, the same properties apply to both methods for the defined stability region of the method. However, the size of the integration step determines whether a given solution is within this region, as can be seen in the following equation for definition of the stability region of TRAPEZ method

$$\frac{x_{n+1}}{x_n} = \frac{1 + \frac{1}{2}z}{1 - \frac{1}{2}z} \quad (14)$$

where $z = hk$, and $\dot{x} = k \cdot x$. It is still necessary to check whether the solution is sufficiently close to the right solution. Thus every numerical step computed by the NNSE method, must be still checked for its accuracy. SPICE implements an error-based time-adaptive stepping algorithm that determines Local Truncation Error $LTE = f(\epsilon)$ [19] via Equation (17). It is based on a function of the currents I and charges Q of the capacitors (or fluxes and voltages of the inductors). It is the maximum of two errors: current error (ϵ_I) and charge error ϵ_Q .

$$\epsilon_I = reltol \cdot \max(|I_i|, |I_{i-1}|) + abstol \quad (15)$$

$$\epsilon_Q = reltol \cdot \frac{\max(|Q_i|, |Q_{i-1}|, chgtol)}{\delta_n} \quad (16)$$

$$\epsilon = \max(\epsilon_I, \epsilon_Q) \quad (17)$$

4. Training of Neural Network Step Estimator Algorithm

Levenberg-Marquardt back propagation was used for NN training [20]. It is designed to minimize the mean-square error between the actual and predicted output of the MPL. Initially, the weights are determined randomly, then the algorithm computes the mean-square error repeatedly for each combination of input and output. Subsequently, the error is propagated back and used to readjust the weights and threshold. This is shown in Figure 2.

For normalization of input values of trained network, the standard normalization algorithm was used

$$x_{i_{norm}} = (x_i - x_{min}) / (x_{max} - x_{min}), i = 1, \dots, L \quad (18)$$

where x_{max} and x_{min} are the highest values before and after normalization, respectively.

Figure 2. Neuron with Error Back Propagation.

5. Implementation

The TensorFlow (TF) library [21] with Keras library [22] was used to implement the NN. Several NN were created with a huge set of training values. It was achieved by simulating

different circuits, which were generated with different constellation of parameters of the electrical components. The parameters were randomly generated within specific range with maintain of their standard functionality. Subsequently, algorithm training was performed for several well known circuits, these were Voltage Controlled CMOS Oscillator, CMOS VCO, Ring Oscillator, MOS Amplifier, JFET Amplifier, and CMOS Multiplier (as also shown in Figures 3 and 4 and Table 1). For all simulations, the initial Transient Analysis was calculated. This was then repeated for randomly changed component parameters with variation of the components ranged from 0% to 2%. Main parameters that varied in circuits were source voltages, resistive, capacitance, and inductive parameters.

Figure 3. Number of Iterations Required to Compute a Given Circuit for Different Types of Integration Methods and Variation of Component Parameters.

Figure 4. Dependence of the Number of Iterations on the Similarity to the Original Simulated Circuit.

Table 1. Time required for pivoting, reordering, and factorization for different sizes of sparse matrix.

	TR	TRP	GR	GRP	GRN 1%	TRN 1%	GRN 5%	TRN 5%	GRN 10%	TRN 10%
CMOS VCO	16,196	17,965	16,477	19,119	16,352	17,201	13,716	16,354	16,216	15,799
Ring Oscillator	9112	9145	11,252	7981	4997	6944	6792	7925	8258	7809
MOS Amplifier	5532	4136	6510	5343	3055	3887	3826	4191	4147	4037
JFET Amplifier	3140	2270	3947	1837	1812	1372	1872	1773	1907	1900
CMOS Multiplier	4012	3322	4079	3996	2096	3346	2805	3149	3355	3408

The NN was trained through the assisted learning process, where pre-computed values of the integration step were presented to the NN. It learned from them the best approximation to the problem. Subsequently, the trained network was used to estimate step size on simulations with component parameters variations from 0% to 25% percent. The setting of the weights was always determined randomly at the beginning of the learning process. The trained NN was implemented in NgSpice as it is shown in the Algorithm 2. Optional parameter `.OPTIONS NNSA=(NN id)` was added to the simulation configuration. The values from the NgSpice program were shared during the simulation to the TF NN algorithm via Unix Named pipes. This step negatively affected the speed of the simulation computation and should be replaced in the final implementation by a native implementation of the NN in the NgSpice code. It should be noted that to avoid non-convergence we implemented a correction algorithm which, after an unsuccessful attempt, calculates the next stopping point according to the standard method (based on LTE).

Algorithm 2 Transient Simulation with NNSE

```

Modified Nodal Analysis
Initial Matrix Composition
Pivoting, Reordering
Classification for NN
for Timeline do
  Linear system (LU Factorization)
  repeat {Nonlinear system (Newton-Raphson) }
    repeat {NI (GEAR/Trapez) }
      NN Estimation of Next Integration Step
      Residual (vector norm)
    until Stopping criteria
  until Stopping criteria
  if not (Convergence) then
    return Convergence problem
  end if
  Optional Pivoting and Reordering
end for

```

6. Simulated Circuits

6.1. Multiplier

This circuit is multiplier which origins from article [23], that contains the circuit diagram, and the model and control parameters are defined in Algorithm 3. It is a low-voltage low-power CMOS four-quadrant microwave multiplier. It use by semi-empirical models of transistors of level 3. As could be seen circuit is designed very precisely and therefore simulation was successful in almost all iterations.

Algorithm 3 Model and Analysis Parameters of Multiplier

```

.MODEL MN NMOS LEVEL = 3 UO = 460.5 TOX =  $1.0 \times 10^{-8}$ 
+TPG = 1 VTO = 0.62 JS =  $1.08 \times 10^{-6}$  XJ = 0.15U
+RS = 417 RSH = 2.73 LD = 0.04U VMAX =  $130 \times 10^3$ 
+NSUB =  $1.71 \times 10^{17}$  PB = 0.761 ETA = 0.00 THETA = 0.129
+PHI = 0.905 GAMMA = 0.69 KAPPA = 0.10 CJ =  $76.4 \times 10^{-5}$ 
+MJ = 0.357 CJSW =  $5.68 \times 10^{-10}$  MJSW = 0.302
+CGSO =  $1.38 \times 10^{-10}$  CGDO =  $1.38 \times 10^{-10}$  CGBO =  $3.45 \times 10^{-10}$ 
+KF =  $3.07 \times 10^{-28}$  AF = 1
.MODEL MP PMOS LEVEL = 3 UO = 100 TOX =  $1.0 \times 10^{-8}$ 
+TPG = 1 VTO = -0.58 JS =  $0.38 \times 10^{-6}$  XJ = 0.10U RS = 886
+RSH = 1.81 LD = 0.03U VMAX =  $113 \times 10^3$  NSUB =  $2.08 \times 10^{17}$ 
+PB = 0.911 ETA = 0.00 THETA = 0.120 PHI = 0.905
+GAMMA = 0.76 KAPPA = 2 CJ =  $85 \times 10^{-5}$  MJ = 0.429
+CJSW =  $4.67 \times 10^{-10}$  MJSW = 0.631 CGSO =  $1.38 \times 10^{-10}$ 
+KF =  $1.08 \times 10^{-29}$  AF = 1 CGDO =  $1.38 \times 10^{-10}$  CGBO =  $3.45 \times 10^{-10}$ 
+KF =  $1.08 \times 10^{-29}$  AF = 1
.TRAN 100P 40N 0 20P

```

6.2. CMOS VCO

This is the largest simulated circuit based on the work published in paper [24]. Basic component models of NMOS and PMOS transistors are completely defined in Algorithm 4.

Algorithm 4 Model Parameters of CMOS VCO

```

.MODEL CMOSN NMOS LEVEL = 3 PHI = 0.600000 TOX =  $2.1200 \times 10^{-8}$  XJ = 0.200000U
+TPG = 1 VTO = 0.7860 DELTA =  $6.9670 \times 10^{-1}$  LD =  $1.6470 \times 10^{-7}$  KP =  $9.6379 \times 10^{-5}$ 
+UO = 591.7 THETA =  $8.1220 \times 10^{-2}$  RSH =  $8.5450 \times 10^1$  GAMMA = 0.5863
+NSUB =  $2.7470 \times 10^{16}$  NFS =  $1.98 \times 10^{12}$  VMAX =  $1.7330 \times 10^5$  ETA =  $4.3680 \times 10^{-2}$ 
+KAPPA =  $1.3960 \times 10^{-1}$  CGDO =  $4.0241 \times 10^{-10}$  CGSO =  $4.0241 \times 10^{-10}$ 
+CGBO =  $3.6144 \times 10^{-10}$  CJ =  $3.8541 \times 10^{-4}$  MJ = 1.1854 CJSW =  $1.3940 \times 10^{-10}$ 
+MJSW = 0.125195 PB = 0.800000

.MODEL CMOSP PMOS LEVEL = 3 PHI = 0.600000 TOX =  $2.1200 \times 10^{-8}$  XJ = 0.200000U
+TPG = -1 VTO = -0.9056 DELTA = 1.5200 LD =  $2.2000 \times 10^{-8}$  KP =  $2.9352 \times 10^{-5}$ 
+UO = 180.2 THETA =  $1.2480 \times 10^{-1}$  RSH =  $1.0470 \times 10^2$  GAMMA = 0.4863
+NSUB =  $1.8900 \times 10^{16}$  NFS =  $3.46 \times 10^{12}$  VMAX =  $3.7320 \times 10^5$  ETA =  $1.6410 \times 10^{-1}$ 
+KAPPA = 9.6940 CGDO =  $5.3752 \times 10^{-11}$  CGSO =  $5.3752 \times 10^{-11}$ 
+CGBO =  $3.3650 \times 10^{-10}$  CJ =  $4.8447 \times 10^{-4}$  MJ = 0.5027 CJSW =  $1.6457 \times 10^{-10}$ 
+MJSW = 0.217168 PB = 0.850000

```

6.3. JFET Amplifier

This is very simple circuit that can be fully defined here, it is defined in Algorithm 5.

Algorithm 5 Full Definition of JFET Amplifier

```

V 1 1 0 DC 0
V 2 2 0 DC 0
J1 2 1 0 J2N3819
.model J2N3819 NJF(Beta = 1.304 m Rd = 1 Rs = 1 Lambda = 2.25 m Vto = -3
+ Is = 33.57f Cgd = 1.6p Pb = 1 Fc = .5 Cgs = 2.414p Kf =  $9.882 \times 10^{-18}$  Af = 1)

```

6.4. Ring Oscillator

The oscillator consists of a chain of odd number of CMOS inverters that generate an oscillation with a period T equal to $2 \cdot N \cdot t_p$, where N is the number of inverters, and t_p

is the propagation delay (2 because each inverter switches twice during one period). The schematic and SPICE models are accessible through web link [25].

6.5. MOS Amplifier

This circuit is MOS transistor amplifier. It originates from NgSpice testing repository and can be downloaded with actual version of NgSpice21. It is modelled by MOS transistors with substrate doping NSUB, gate-source overlap capacitor CGSO, gate-drain overlap capacitor CGDO, surface state density NSS and surface mobility UO, oxide thickness TOX, lateral diffusion LD, and critical field exponent for mobility degradation UEXP. All the analysis and model parameters are defined in Algorithm 6.

Algorithm 6 Analysis and Model Parameters of MOS Amplifier

```
.OPTIONS ABSTOL = 10n VNTOL = 10n NOACCTO
.TRAN 0.1us 10us
.MODEL m NMOS NSUB =  $2.2 \times 10^{15}$  UO = 575 UCRIT = 49k
+UEXP = 0.1 TOX = 0.11u XJ = 2.95u LEVEL = 2
+LEVEL = 2 CGSO = 1.5n CGDO = 1.5n CBD = 4.5f
+CBS = 4.5f LD = 2.4485u NSS =  $3.2 \times 10^{10}$ 
+KP =  $2 \times 10^{-5}$  PHI = 0.6
```

7. Results

Figure 3 shows a comparison of the individual methods, namely the standard TRAPEZ method and GEAR method. In addition, the predictor-corrector scheme was also used for circuits. The graph shows the compared values for the algorithms used on circuits with component parameter variances of 1%, 5%, and 10% from the original trained network. It can be seen that the NNSE augmented simulations perform well when the circuit similarity is high, i.e., with a variation of circuit variables up to a maximum of 5% percent. In those cases NNSE helped in most cases to reduce the number of iterations. The problem arises when the variance is above 10% and the number of inconsistencies starts to grow up to the point where the standard stepping mechanism takes over. For very small circuit changes, the NNs performed noticeably better than standard algorithms that had to adaptively calculate an estimated integration step. In the case of the NN, this step was dropped and the network only quantified the error of the step change for the non-convergence case. The NNSE performed very well up to the variation of circuit quantities of 5%. Beyond this limit, problems began to manifest when the integration step had to be recalculated because of rejection by LTE or non-convergency. Above about 20%, the circuit definition was so different that the algorithm became unstable and step rejection algorithm had to start correcting the NI by repeatedly reducing the size of the integration step. The values in the table characterize the number of iterations required to complete the transient simulation of a given circuit. For NN simulations, these are then integer counts of the average number of iterations for the circuit. In the table, TR denotes the TRANS method, TRP denotes TRANS with predictor-corrector, and GR is the GEAR method. Sufficing with the letter N indicates NNSE step estimation. The percentages then denote the variance of the component parameters in the circuit.

The bolded values highlight the algorithm that achieved the best result, i.e., the lowest number of iterations needed to achieve the same result. However, this does not mean that in other cases the NNSE algorithm does not also perform well/better than the standard algorithm. For the CMOS Multiplier example, the algorithm achieved virtually identical or better results than the standard algorithms in all cases.

Figure 3 shows the dependence of the minimized number of iteration steps required to compute a given circuit simulation using NNSE on the percentage deviation from the original circuit parameters. A trend can clearly be seen that our trained NNs cease to be relevant to the properties of the simulation of a given circuit from a certain point onwards. A possible solution would be to generate a larger neural network for a larger

input dataset. However, this did not prove practical as the larger set degraded the overall accuracy of the NN and thus its performances against standard methods at low variations of circuit parameters.

The stability of the NI step computation can be seen from Figure 5. It shows the progression of several steps using the original algorithm shown in the figure by the dashed line, and the neural network based algorithm (solid line). This plot was generated for the variance of the circuit variables within 1 percent of the original circuit. There are noticeable differences from the original magnitude, but of such an order of magnitude that it does not yet affect the calculation process. What is important to note, however, is that the NNSA lacks the divergence of the original method. Hence, the NNSA curve is one value shorter than the original calculation process.

Figure 5. Comparison of Local Integration Error for Standard method and NNSA.

8. Computational Setup

All calculations were carried out by using a desktop computer with an Intel Core i9-9900K (3.60 GHz $\times$ 16), 31.3 GB of RAM memory and a NVIDIA GeForce RTX 2080 Ti graphics card with 11 GB memory.

9. Conclusions

We presented a NI step prediction mechanism NNSP based on NN. This network was built to define the integration of a given circuit over a range of circuit parameters and components used. The goal of the new algorithm was to estimate the integration step size based on the trained NN so as to minimize as much as possible the number of non-convergences and restarts of the NI due to incorrect step size. The proposed method was tested in NgSpice, an open-source copy of SPICE. The algorithm has been implemented as a companion to the standard algorithms for computing numerical integration in SPICE, i.e. the TRAPEZ and GEAR methods. The original step computation was used to train the NN, and the new simulation option parameter was used to specify which trained NN should be used. The method was shown to be able to precompute step sizes for certain circuits if correctly classified. In these cases, it then allowed reducing the number of iterations. Each circuit had a specially trained network. For real use, the process needs to be supplemented with an algorithm or with another NN which would be used to correctly classify the circuit according to the components involved and then select the correct NN. Training a generic network that could then be used for all types of simulation problems is certainly also up for discussion.

Author Contributions: Neural network step estimator algorithm, analyzing current status of the NgSpice simulator, algorithm for training of neural network step estimator, implementation of the algorithm into NgSpice, creation of graphics D.Č.; artificial neural network arrangement, algorithms for numerical integration, comparing and testing methods, J.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Czech Science Foundation under the grant No. GA20-26849S.

Acknowledgments: This paper has been supported by the Czech Science Foundation under the grant No. GA20-26849S.

Conflicts of Interest: The authors declare no conflict of interest. The funder had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Levin, A.; Lischinski, D.; Weiss, Y. Colorization using optimization. In *ACM SIGGRAPH 2004 Papers*; ACM: New York, NY, USA, 2004; pp. 689–694.
2. Wu, Z.; Watts, O.; King, S. Merlin: An Open Source Neural Network Speech Synthesis System. In *Proceedings of the 9th ISCA Workshop on Speech Synthesis Workshop (SSW 9)*, Sunnyvale, CA, USA, 13–15 September 2016; pp. 202–207.
3. Mukalov, P.; Zelinskyi, O.; Levkovych, R.; Tarnavskiy, P.; Pylyp, A.; Shakhovska, N. Development of System for Auto-Tagging Articles, Based on Neural Network. 2019. Available online: <http://ceur-ws.org/Vol-2362/paper10.pdf> (accessed on 25 November 2021).
4. Nguyen, X.A.; Ljuhar, D.; Pacilli, M.; Nataraja, R.M.; Chauhan, S. Surgical skill levels: Classification and analysis using deep neural network model and motion signals. *Comput. Methods Programs Biomed.* **2019**, *177*, 1–8. [CrossRef] [PubMed]
5. Vladimirescu, A. SPICE-the third decade. In *Proceedings of the Bipolar Circuits and Technology Meeting*, Minneapolis, MN, USA, 17–18 September 1990. [CrossRef]
6. Zhe-Zhao, Z.; Yao-Nan, W.; Hui, W. Numerical integration based on a neural network algorithm. *Comput. Sci. Eng.* **2006**, *8*, 42–48. [CrossRef]
7. Uddin, M.; Ullah, M. A Modified Predictor-Corrector Formula For Solving Ordinary Differential Equation Of First Order and First Degree. *IOSR J. Math.* **2013**, *5*, 21–26. [CrossRef]
8. Tripodi, E.; Musolino, A.; Rizzo, R.; Raugi, M. A new predictor–corrector approach for the numerical integration of coupled electromechanical equations. *Int. J. Numer. Methods Eng.* **2016**, *105*, 261–285. [CrossRef]
9. Acary, V.; Bonnefon, O.; Brogliato, B. Time-Stepping Numerical Simulation of Switched Circuits Within the Nonsmooth Dynamical Systems Approach. *Comput.-Aided Des. Integr. Circuits Syst.* **2010**, *29*, 1042–1055. [CrossRef]
10. Ferreira, D.; Oliveira, J.; Pedro, J. A Novel Time-Domain CAD Technique Based on Automatic Time-Slot Division for the Numerical Simulation of Highly Nonlinear RF Circuits. *Microw. Theory Tech.* **2014**, *62*, 18–27. [CrossRef]
11. Ding, Z.; Li, L.; Hu, Y. A modified precise integration method for transient dynamic analysis in structural systems with multiple damping models. *Mech. Syst. Signal Process.* **2018**, *98*, 613–633. [CrossRef]
12. Nenzi, P.; Vogt, H. Ngspice Users Manual Version 23. 2011. Available online: <https://src.fedoraproject.org/repo/extras/ngspice/ngspice23-manual.pdf/eb0d68eb463a41a0571757a00a5b9f9d/ngspice23-manual.pdf> (accessed on 25 November 2021).
13. Nichols, K.; Kazmierski, T.; Zwolinski, M.; Brown, A. Overview of SPICE-like circuit simulation algorithms. *Circuits Devices Syst. IEE Proc.* **1994**, *141*, 242–250. [CrossRef]
14. Demiryurek, O.; Yildiz, A. Modified nodal analysis formulation of operational transconductance amplifier. In *Proceedings of the 5th European Conference on Circuits and Systems for Communications (ECCSC'10)*, Belgrade, Serbia, 23–25 November 2010; pp. 173–176.
15. Gubian, P.; Zanella, M. Stability properties of integration methods in SPICE transient analysis. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, Singapore, 11–14 June 1991.
16. Gear, C.W. *Numerical Integration of Stiffy Ordinary Equations*; Technical Report 221; Department of Computer Science, University of Illinois: Urbana, IL, USA, 1967.
17. Almeida, L.B. C1. 2 Multilayer Perceptrons. In *Handbook of Neural Computation C*; Elsevier: Amsterdam, The Netherlands, 1997.
18. Brownlee, J. A gentle introduction to the rectified linear unit (ReLU). *Mach. Learn. Mastery* **2019**, *6*, 9.
19. Gupta, G.K.; Sacks-davis, R.; Tischer, P.E. A Review of Recent Developments in Solving ODEs. *ACM Comput. Surv.* **1985**, *17*, 4–47. [CrossRef]
20. Sapna, S.; Tamilarasi, A.; Kumar, M.P. Backpropagation learning algorithm based on Levenberg Marquardt Algorithm. *Comp. Sci. Inform. Technol.* **2012**, *2*, 393–398.
21. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015. Available online: <https://github.com/tensorflow/tensorflow> (accessed on 25 November 2021).
22. Chollet, F. Keras–GitHub. 2015. Available online: <https://github.com/fchollet/keras> (accessed on 25 November 2021).

-
23. Dobes, J. Advanced types of the sensitivity analysis in frequency and time domains. *Int. J. Electron. Commun.* **2009**, *63*, 52–64. [[CrossRef](#)]
 24. Kumar, A.; Tiwari, K. Wide Tuning Range CMOS VCO. *Int. J. Eng. Manuf.* **2017**, *7*, 31–38. [[CrossRef](#)]
 25. Baker, J. Ring Oscillator with odd number of CMOS Inverters. Available online: <https://www.youspice.com/spiceprojects/spice-simulation-projects/signal-generation-circuits-spice-simulation-projects/ring-oscillator-with-odd-number-of-cmos-inverters/> (accessed on 25 November 2021).