intusoft (310) 833-0710

Personal Computer Circuit Design Tools

N E W S L E T T E R

New IsSpice/286 Breaks the DOS 640K Barrier

Intusoft announces the end to the "Memory Limit Exceeded" error message. In the tradition of IsSpice/386, the protected mode version of IsSpice for 80386 machines, ***intusoft*** announces IsSpice/286. IsSpice/286 is a version of Spice (Berkeley compatible) that runs in protected mode under DOS 3.1 and up. It will allow any 80286 based PC to utilize Extended memory to run Spice simulations. With IsSpice/286 you can simulate circuits with over 2000 components with as little as 2 megabytes of extended memory. IsSpice/286 runs about 20% slower than IsSpice v1.41. So in order to provide you with the most efficient simulation possible, ***intusoft*** is including the IsSpice version 1.41 (real mode) Spice simulator with each new order of IsSpice/286. The combination of the two programs will allow users to run simulations under 200 nodes at top speed without having to pay for the protected modes' complex addressing scheme. But when a large simulation capability is needed for an ASIC design or a lengthy switching power supply simulation, users can then call upon the power of IsSpice/286.

In This Issue

Using SPICE for Neural Network Development

Neural networks, or nonlinear adaptive control systems, are emerging as a viable new technology. Back in November, 1986 [1] we introduced a set of digital elements using threshold logic techniques to improve the performance of SPICE for digital simulation. It turns out that one class of neural networks, based on the Perceptron, are in fact a modification of threshold logic theory. We are developing the modeling extensions necessary to simulate these networks with SPICE. Before discussing the theory and modeling techniques, however, an exploration of potential applications for these networks will provide a road map for our simulation goals.

New Opportunities for Innovation

First, the theory is still developing. Second, there are more questions than answers. Some simple networks, when extended to solve more complex problems, will use more computing resources than are available. These problems are often solved using network topologies based on heuristic principals. Heuristics insights are drawn from Biology, Mathematics and Engineering. In Biology, the learning characteristics for simple action-reaction processes have been measured and the basic anatomy is known. In Mathematics and Engineering, the models for neurons have been developed along with some stability criteria. Many heuristic networks and learning algorithms are being pursued; for example, Sejnowski [2] describes a 2 layer perceptron with 120 hidden nodes that was trained to transcribe text phonetically into speech. Simulation tools developed along these lines can help shape the theory and develop better network topologies.

Another direction of research leads to the practical application of these networks. Current artificial intelligence, AI, technology uses knowledge based expert systems. This type of system can only be applied to a static technology because the decisions are rule based. The rules are cast by picking the brains of human experts. These problems in AI systems can be solved using neural nets. First, neural nets can be trained by example, thus developing their own rules and adapting to change. Second, the training may not require the interface between a team of programmers and a human expert.

Putting Neural Nets to Work

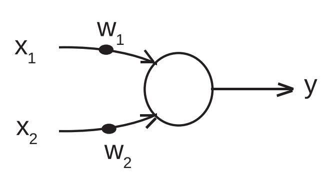
As our society produces more complex high technology products, the people who understand these products diminishes, first, to the point of not being able to repair or maintain the products and second, to the point of not even getting a new product through a manufacturer's production line. Built-in test and redundancy are being used to solve some of these problems. Perhaps neural network technology can be developed along with simulation to produce intelligent test systems. To accomplish this, there must be a neural network capable of handling the problem and a number of training sets that can be applied to the network before production components are to be evaluated. SPICE can provide the initial training sets and also assist in network development, both at the circuit level and the theoretical level.

The Perceptron,

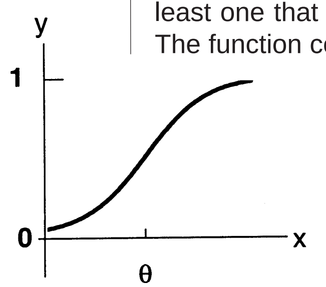
an Analog of Biological Neural Activity

Shown below is the basic threshold logic element. Two signals, x_1 and x_2, are multiplied by weights w_1 and w_2 and the sum of the result is compared with a threshold to form the y output. Logic states are all taken as 0 or 1.

$$\begin{aligned} y &= 1, \text{ if } w_1x_1 + w_2x_2 > y_t \\ &= 0, \text{ if } w_1x_1 + w_2x_2 < y_t \end{aligned}$$



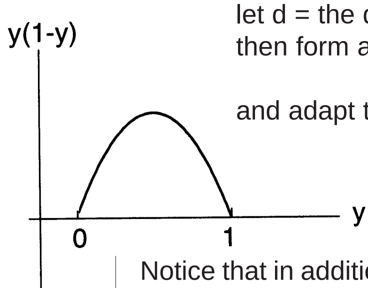
To make this an adaptive network, a set of training inputs is applied sequentially. The weights, w_1 and w_2, are changed slightly whenever the result is in error and when they have an output contribution, that is, w_1 is changed only if x_1 is 1. It can be seen that this approach yields separating functions that are almost wrong because the adaptive correction stops within a small neighborhood of the previous wrong answer. An improvement is made to this network by replacing the hard limiting threshold with a function that produces an output with a proportional error characteristic. It turns out that a continuous function (at least one that is continuously differentiable) is preferred. The function commonly used is given by:



$$y = \frac{1}{1 + e^{-(x-\theta)}}$$

where y is the weighted sum of inputs and θ is the threshold.

The difference between the desired output and the actual output is then fed back to control the weights as follows:



let d = the desired output,
then form an error, $\delta = y(1-y)(y-d)$

and adapt the weights, $w_j(t + 1) = w_j(t) + \eta\delta x_j$,

where η is a gain constant and t is time

Notice that in addition to the gain shaping of the threshold limiter, the function y(1-y) that was placed in the adaptive loop has high gain only in the threshold region. Instabilities caused by high gain near threshold will tend to kick the system out to the more stable low gain regions. This will tend to limit chaotic results, however, final solutions can still be sensitive to initial conditions.

Adding Layers with Back Propogation

When additional layers of perceptrons are connected, the intermediate layers use the average δ of the next layer, taken backward from the output, to form their weight computations. This is the back propagation algorithm of Rumelhart [3]. A 3 layer perceptron is sketched in Figure 1. The advantage of adding layers lies in the ability to separate complex shapes; for example the points enclosed by an image of the letter 'O' can be classified using 3 layers.

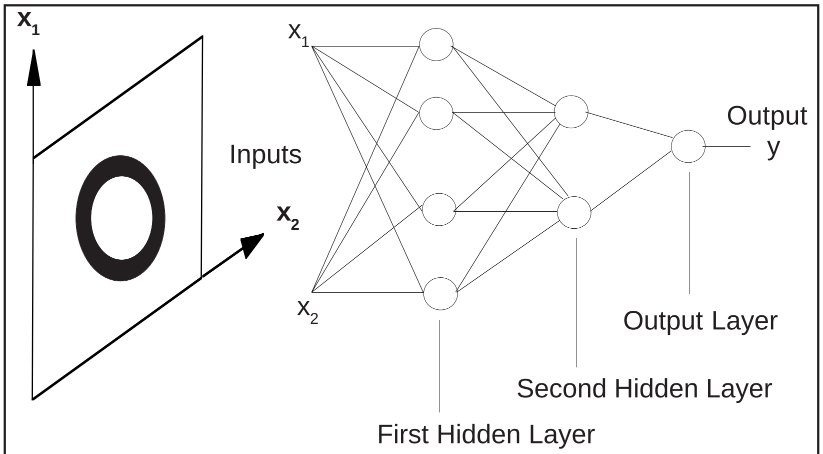


Figure 1, A 3 Layer Perceptron classifies inputs that belong in the pattern 'O'.

Building SPICE Models

In order to make SPICE models for the perceptron, it is necessary to break the computation into blocks that can be easily expanded to include multiple layers and many nodes within each layer. The block diagram shown in Figure 2 shows the interconnect or Axon block. Inputs are voltages and outputs are currents. This allows the forward and back propagation summations to take place by connecting wires. The unit time delay represents the recursive iteration, its formulation will be discussed later. The elements GA and GB are used as multipliers.

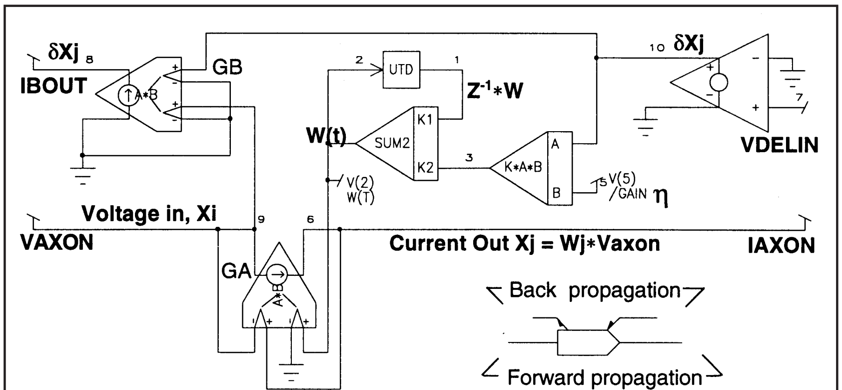


Figure 2, An Axon block diagram and its symbolic representation

Building SPICE Models *(Continued)*

The second building block shown in figure 3 is called a Cell. It performs the threshold logic summation and provides the threshold function. The back propagation window, y(1-y), is formed in the top loop. Elements shown in the block diagrams are all found in the ***intusoft*** System model libraries. The representations can be streamlined by flattening the subcircuits and removing duplicate buffering parts. The flattened SPICE listing is given in Table 1.

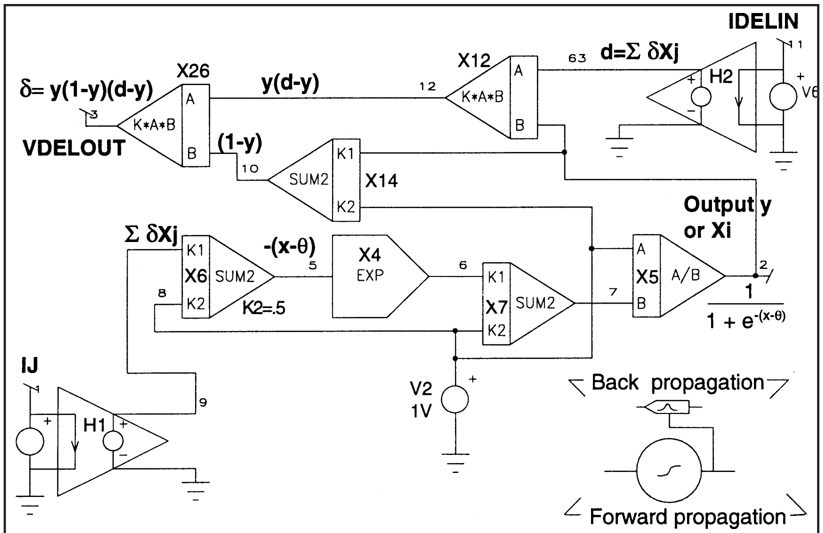


Figure 3, The Cell performs the threshold logic summation and the back propagation window function

Table 1, SPICE Listings for Neural Net Elements

```
********
.SUBCKT CELL 1 2 11 3
* 2 Output y or Xi
* 1 IJ - Current Summation
* 11 IDELIN - Back Propagation In
* 3 VDELOUT - Back Propagation Out
*Resistors to make Spice Happy, 2
*connections per node
RX2 2 0 1K
RX9 9 0 1K
RX7 7 0 1K
RX8 8 0 1K
RX63 63 0 1K
RX12 12 0 1K
RX5 5 0 1K
RX6 6 0 1K
RX10 10 0 1K
RX3 3 0 1K
X5 8 7 2 DIVIDE
EX6 5 0 POLY(2) 9 0 8 0 0 -1 .5
EX7 7 0 POLY(2) 6 0 8 0 0 1 1
V2 8 0 1
H1 9 0 VJ 1
VJ 1 0
*Multiplier below
EX12 12 0 POLY(2) 63 0 2 0 0 0 0 0 1
EX14 10 0 POLY(2) 2 0 8 0 0 -1 1
H2 63 0 V6 1
V6 11 0
EX26 3 0 POLY(2) 10 0 12 0 0 0 0 0 1
X4 5 6 EXP
.ENDS
*******
.SUBCKT AXON 9 6 7 8 5 {W=.01}
*W=.01 Default Value for Weight
*5 Gain, η
*8 IBOUT - Back Propagation Output
*9 VAXON - Voltage In
*6 IAXON - Current Out
*7 VDELIN - Back Propagation Input
** Z Transform Unit Time Delay **
ETD1 30 0 2 0 1
RTI 30 0 1K
T1 30 0 1 0 Z0=1 TD=1
RTD1 1 0 1
**
RX3 3 0 1K
RX2 2 0 1K
RX10 10 0 1K
*Summer below, W is the initial
*condition of the weight passed
*into the subcircuit as a parameter
*and evaluated using PreSpice
EX7 2 0 POLY(2) 1 0 3 0 {W} 1 1
GA 9 6 POLY(2) 2 0 6 9 0 0 0 0 1
GB 0 8 POLY(2) 10 0 9 0 0 0 0 0 1
E1 10 0 7 0 1
EX3 3 0 POLY(2) 10 0 5 0 0 0 0 0 1
.ENDS
*******
.SUBCKT VTOI 1 2 3
* I(3) = -V(1) + V(2)
G 0 3 POLY(2) 1 0 2 0 0 -1 1
.ENDS
******
```

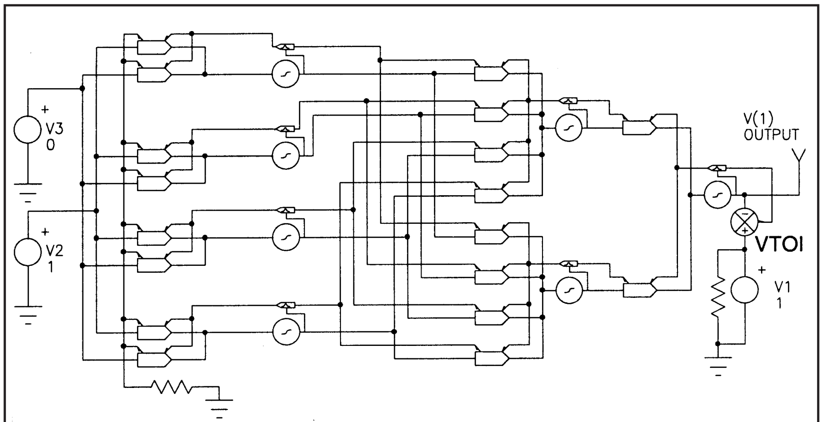


Figure 4, A 3 layer perceptron using the new SPICE symbols

Next, Figure 4 shows how the blocks are put together to make the 3 layer Perceptron shown in Figure 1. A simple summing symbol was added to make the training set comparison.The unit time delay is used in the Axon block to form the recursive summation. An alternate implementation will be shown in the next newsletter using switched capacitors. Both implementations need a method for initialization of the weights and the introduction of training sets. Our next newsletter will illustrate these techniques and add several examples. In the meantime, you can look through Lippman [4] for a survey of some other neural net algorithms or purchase the book by Rumelhart [3], from MIT press, volume 1 and volume 2. A set of software sample problems is also available.

Bibliography

[1] Simulating With Spice
Meares, L.G.; Hymowitz, C.E.
Intusoft, 1988 pp. 4-15 to 4-20
Originally published Nov, 1986 Intusoft Newsletter

[2] NetTalk: A Parallel Network That Learns to Read Aloud
Sejnowski,T.; Rosenberg,C.R.
Johns Hopkins Univ. Technical Report JHU/EECS-86/01, 1986

[3] Learning Internal Representations By Error Propagation
Rumelhart,D.E.; Hinton,G.E.; Williams,R.J.
Parallel Distributed Processing: Exploration in the Microstructure of Cognition, Vol 1, Foundations, MIT Press, 1986

[4] An Introduction to Computing With Neural Nets
Lippmann, R.P.
IEEE ASSP Magazine pp. 4-21 April, 1987

The Versatility of

SPICE is an excellent tool for performing a broad spectrum of system analyses. As a design takes shape, these analyses can be used to form or verify system specifications. When the design matures, the various system elements can be replaced one at a time by actual circuitry and the system performance re-evaluated stage by stage. This process would create a superior design with individual portions fitting more smoothly into the overall system while providing a better understanding of the design in general. The wide variety of analog computer functions that are found in the PRESPICE and SPICENET programs can be applied to a vast array of applications.

The schematic below describes a pinewood toy car race. Of interest is the car's acceleration, velocity, and distance traveled under the effects of gravity and various coefficients of aerodynamic friction. The simulation was actually prompted by a young boy's request to find out which one of his two race cars would be a better bet to win a local pinewood derby.

The race consisted of releasing the car, initially at rest, from the top of a hill (incline 45° X axis 8', Y axis 8'). At the X axis distance of 8', the downward incline leveled

Analog Computer Models

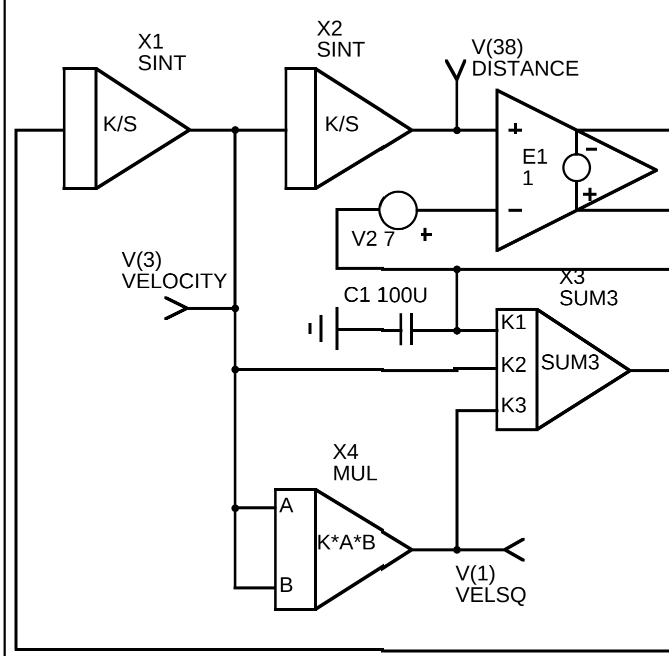
These elements have a wide variety of uses as we have seen in the Neural Network article and above. With SPICENET (schematic entry) and PRESPICE (model libraries), the ease of use of these models is greatly facilitated.

RACE Circuit
Top Level Netlist

```
*INCLUDE DIGITAL.LIB
*INCLUDE SYS.LIB
.TRAN .1 5 UIC
*ALIAS  V(38)=VDISTANCE
*ALIAS  V(3)=VELOCITY
*ALIAS  V(5)=VACCEL
*ALIAS  V(7)=VGRAVITY
.PRINT TRAN  V(38)  V(3)  V(5)  V(37)
.PRINT TRAN  V(7)  V(2)  V(1) V(4)
X2 3 38 SINT  {K=1}
X3 7 3 1 5 SUM3 {K1=7 K2=-.21M
+ K3=-2.1M}
X4 3 3 1 MUL  {K=1}
V2 4 7 7
RV2 4 0 1MEG
E1 0 2 38 4 1
RE1 2 0 1MEG
X7 2 0 37 LIMITER
R1 7 37 1K
R37 37 0 1MEG
C1 7 0 100U IC=1
X1 5 3 SINT  {K=3.2 }
*SCALED TIMES 1/MASS
.END
```

Subcircuits

```
* INTEGRATOR
.SUBCKT SINT 1 2
*PARAMS ARE GAIN={K}
RIN 1 0 1E12
E1 3 0 0 1 {K}
C1 2 4 1U IC=0
R1 3 4 1MEG
E2 2 0 0 4 {1E6}
.ENDS
*************
.SUBCKT LIMITER 1 2 3
*              + - OUT
* LIMITS OUTPUT TO
* 0 OR 1 VOLT
B3 3 0 V=V(1) - V(2) < 0 ? 0 :
+ V(1) - V(2) > 1 ? 1 :
+ V(1) - V(2)
.ENDS
*************
```

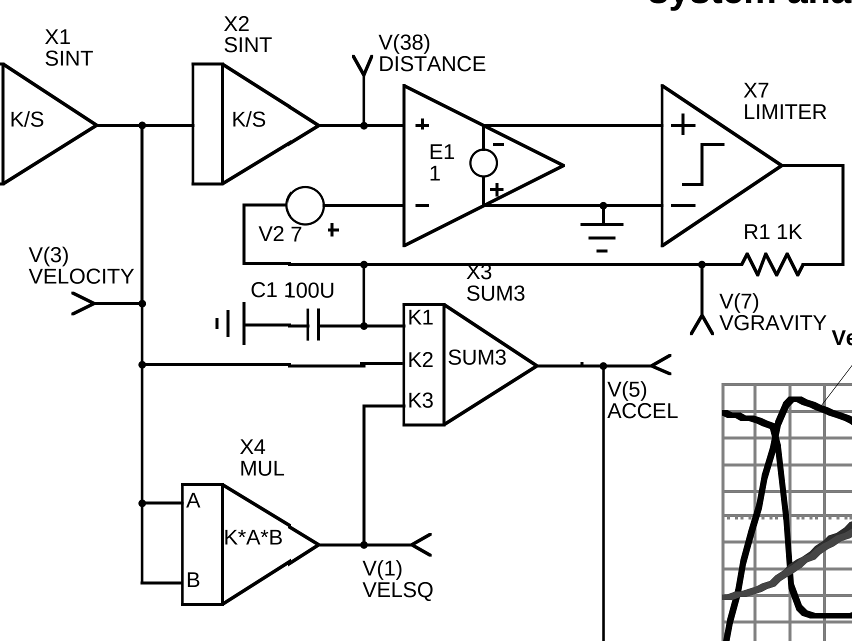


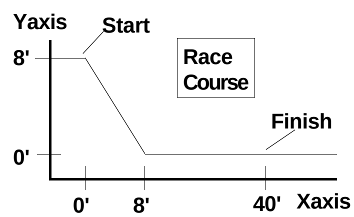
IsSpice - A Toy Car Race

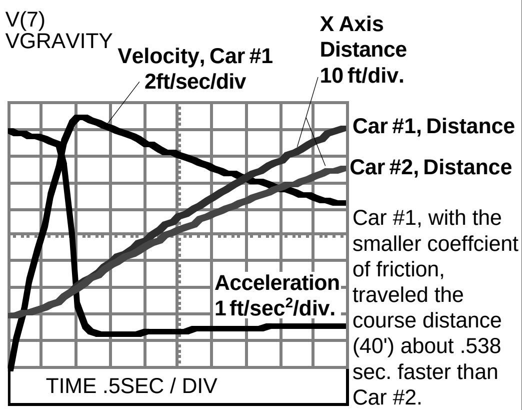
off. The finish line was at an X axis of distance of 40' from the start line. The objective was to determine which of the two cars, each with a different aerodynamic coefficient of friction, would negotiate the course faster.

The schematic shows the interconnection of the PreSpice System library elements and the results. At E1's input, the x axis distance traveled, V(38), is compared to V2 (7 volts) plus the limiter output (1 volt at the start). When V(4) reaches 8, corresponding to the bottom of the hill, the limiter output will go from 1 to 0 and thus remove the effects of gravity. The equation Accel = $7G + K_1V + K_2V^2$ is used to derive the acceleration. G is the effect of gravity, and K_1 and K_2 are coefficients of friction. The velocity and the distance traveled are derived from the acceleration by the integrator elements. R1 and C1 are needed to stabilize the high gain of the limiter element. By altering the coefficients of friction we were able to simulate the performance of each car. The simulation shows, as expected, that the car with the smaller coefficient of friction traveled course distance in a shorter amount of time; about .538 seconds. Oh, and about the race, our young engineer lost. It seems that he was disqualified for obtaining outside assistance.

Spice is an effective simulation tool for system analysis







Car #1, with the smaller coeffcient of friction, traveled the course distance (40') about .538 sec. faster than Car #2.