Remaking the PDP Network:
A Three Node Solution to the XOR Problem

Marshall W. Van Alstyne
Final Project, May 1985

Abstract: This paper details a working solution to the longstanding XOR problem as well as several examples of its use in constructing larger networks. Using a specific instance of the generalized delta rule, it solves certain well defined problems in record time. This solution is not without its faults but these almost certainly are outweighed by its advantages.

In 1969, Minsky and Papert published a paper enumerating many of the flaws in simple perceptron-like neural models. They discussed one still pervasive problem: the "exclusive or." Any model performing this operation needs to produce a single positive answer in response to either of two positive inputs but not to both at the same time. It has long since been argued that if a model cannot solve even this simple problem then it cannot accurately represent the real world and it would be a poor candidate for constructing larger networks of which the XOR is a necessary subcomponent. Intuitively the model should consist of a mere three nodes: two for input and one for the requisite output. Abundant mathematical proof exists, however, as to why the traditional monotonically increasing (or decreasing) formalisms such as summation, multiplication, and the logistic function cannot solve the problem in fewer than four nodes. In this paper, I propose a solution using a sinusoidal function which, because of its symmetry and its cycling, implements a very elegant demonstration of XORing.

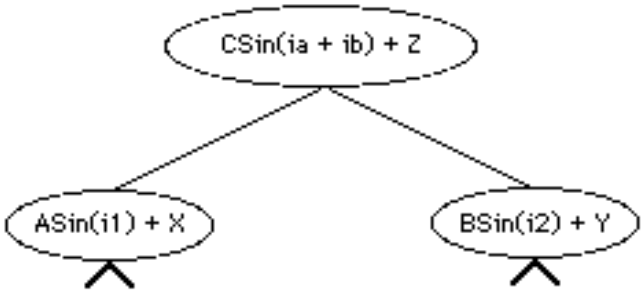
I arrived at this solution not by concentrating on the connections between nodes in a neural network, as has been the tradition, but by concentrating on the output function which operates on node input before passing it on to other nodes. Most learning in this formalism occurs in the node itself not in its connections. One might best describe each node as a "sigma sigma" unit, a name derived from the Greek character used for summation. Each node

first sums all its inputs before operating on them as a set and adding the results. This contrasts with a sigma pi unit which first multiplies its inputs together before operating on them.

Definition of the System:

In addition to a set of weights, this system uses two additional variables which I term a connection bias and a resting bias. The connection bias serves much the same purpose as a traditional connection weight therefore in much of the treatment that follows connection weights will be ignored. Location distinguishes a traditional weight from a connectionbias. A weight represents a property of a connection *between* nodes whereas a connection bias is an intrinsic property of a *single* node. A weight operates on the input passing from one node to another while a connection bias multiplies some function over the sum of all inputs. Connection biases imply a weaker kind of link between nodes than do weights, hence their classification as a "bias." Resting biases, on the other hand, function altogether differently. One may consider the value of a resting bias as the level of activation of any node which has not yet received any input. This constitutes a predisposition on the part of every node to send out some level of information at all times. Such a bias may serve to activate or inhibit other nodes independent of external information, thereby localizing a good deal of system knowledge. A resting bias may change, however, as some function of its input. An unchanged bias merely implies that at no time does a node sit purposeless and inactive waiting for input to process. As a result, networks with resting biases are better equipped to process input according to context. If one series of input leaves behind an information residue stored in a bias, this same information becomes available to help process subsequent input -- a very useful property for disambiguation. Lastly I depart from more traditional network formalisms by substituting the sine function for monotonic functions such as multiplication, addition, and exponentiation. Choosing this as an output function offers several advantages evident in the simulations that follow.

The following graphic depicts a figurative representation for a remodeled network composed of resting biases, connection biases, and the sine function.



In this instance A, B, and C all represent connection biases while X, Y, and Z represent the resting biases. The inputs, ia and ib, to the top node refer to the net total output from the bottom left and the bottom right nodes respectively. A proof that this configuration will compute the XOR function is relatively straightforward. Given a table of inputs and outputs:

Input:	**1 0**	**Output:**	**1**
	0 1		**1**
	0 0		**0**
	1 1		**0**

the net total output of node C is given by the formula:

CSin (ASin (i1) + X + BSin (i2) + Y) + Z = Output

Plugging in the values of 0 0 and 1 1 for input and setting the equations equal results in:

Sin (X + Y) = Sin (X + Y + ASin (1) + BSin (1))

by subtracting the Zs and canceling out the Cs. Setting A = -B to cancel out the sine terms on the right hand side offers the most obvious solution. This yields an identity which is always true. Now substituting in the values of 0 1 and 1 0 and setting the equations equal will produce:

Sin (X + Y + BSin (1)) = Sin (X + Y + ASin (1))

or

Sin (X + Y - ASin (1)) = Sin (X + Y + ASin (1))

Since the sine function is symmetric about $\pi/2$, one can arbitrarily set the sum of (x + Y) to $\pi/2$ and guarantee that the above equation always has a solution regardless of the magnitude of the coefficient A or the exact values of X and Y. This configuration only imposes the restrictions that the input connection biases must be equal in magnitude but opposite in sign and that the resting biases must sum to $\pi/2$. I add one further restriction that no connection bias may be set to zero lest its input never influence its output. As a matter of fact, however, even these somewhat arbitrary definitions restrict unnecessarily and I use them purely for illustration. These equations are underdetermined, i.e., there are fewer equations than variables given. An alternative solution sets X, Y, and Z to zero, ASin(1) = $\pi/4$, BSin(1) = $(3/4)\pi$, and C to $1/\mathrm{Sin}(\pi/4)$.

Derivation of the Learning Rule:

Much of the power of a perceptron-like unit rests in its ability to learn new patterns and to modify its output. Unless the above formalism can match the learning performance of existing systems it becomes little more than a clever trick
for solving the XOR problem. Fortunately, this is not the case. If the error is given by:

equation 1: $$E_p = (1/2) \sum_j (t_{pj} - o_{pj})^2$$

i.e. one half the sum of the squares of the differences between the target output **t** and the actual output **o** for node **j** given the input pattern **p**. Let the connection bias be **ß**. The contribution of **ß** to the error is then given by the following partial differentials:

$$\text{equation 2:} \quad (\partial E_p / \partial O_{pk})(\partial O_{pk} / \partial ß_k)$$

$$\text{equation 3:} \quad (\partial E_p / \partial O_{pk})(\partial O_{pk} / \partial net_{pk})(\partial net_{pk} / \partial ß_j)$$

where, in equation 2, ∂O_{pk} represents the partial derivative of the output of a node k for input pattern p, and where $\partial ß_k$ represents the change of that output with respect to changes in the connection bias. This formula describes the change in error of the output for an output node. For nodes feeding into node k, e.g. node j, the change in error with respect to $ß_j$ is given by equation 3. Here ∂net_{pk} equals the net input to node k since one needs to know the net change in this input as a function of what portion of that input gets fed in from node j.

Given the sinusoidal function described above, a learning algorithm should therefore modify the connection biases for output nodes according to:

$$\Delta ß_k = (O_k - t_k)\text{Sin}(net_{pk})$$

and for non-output nodes one level removed, should modify the connection biases according to:

$$\Delta ß_j = (O_k - t_k) ß_k \text{ Cos}(net_{pk}) \text{ Sin}(net_{pj})$$

Basically, connection node biases arbitrarily far removed may be modified according to this second formula. To compute the error for such a node, multiply the gross error by the connection biases and the cosine terms for all nodes that the given node feeds into and finally multiply by the node's own sine term. In actuality, this is a specific adaptation of the generalized delta rule for learning which Rumelhart, Hinton, and Williams developed in a paper on error propagation.[3]

Related Work:

Rumelhart et. al. developed the generalized delta rule to help apportion the error among input, output, and hidden nodes in large networks of several possible layers. In their system, input nodes accept the environment's external stimuli (the ones and zeros fed into an XOR network for example), output nodes produce the final result, and hidden nodes encode internal representations of a system's processing

knowledge. They are "hidden" in the sense that they do not interact with the environment in any direct way as do input and output nodes. Their paper, entitled "Learning Internal Representations by Error Propagation," attacks many of the same problems as those addressed here. Consequently, a comparison of their results with results obtained here will illustrate the comparative advantages of each type of network formalism.

Briefly, they take the position that:

> Whenever the representation provided by the outside world is such that the similarity structure of the input and output patterns are very different, a network without internal representations (i.e. a network without hidden units) will be unable to perform the necessary mappings. A classic example of this case is the exclusive-or (XOR) problem ...

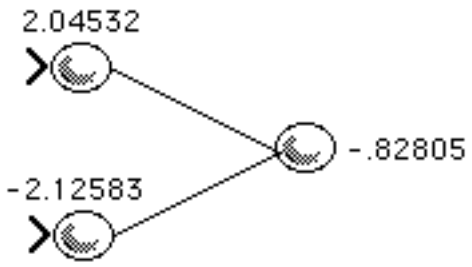
Such a system seeks to permute its input into some more readily usable form of internal representations which it can better recognize and manipulate. The computationally confusing aspect of the XOR problem, as they point out, results from the requirement that the most similar input must generate the most dissimilar output. For example 0 1 and 1 0 both generate a 1 although their dot product is 0 meaning they are exactly orthogonal. The input vector 1 1 which generates a 0 , on the other hand, has a non-zero dot product for both of these other inputs indicating a certain degree of parallelism. One expects like input to produce like output; here, they do not.

Using hidden units to recode the problem, Rumelhart et. al. solve the XOR problem with four nodes, as well as several more complicated problems of which the XOR network is a subcomponent. A network using hidden nodes wields considerable computational power especially in problem domains which seem to require some form of internal representation albeit not necessarily an XOR representation. Those areas common to both papers include the XOR problem, a parity problem, and a negation problem. One area they treat quite well and which I have so far failed to solve neatly without hidden nodes is the symmetry problem. A network to recognize palindromes, for example, notices both parity (i.e. evenness or oddness) as well as location symmetry. If one allows for additional sets of connections to hidden nodes, the problem becomes solvable. This would support an argument to integrate the two systems under certain circumstances, but as will be seen shortly, with additional nodes comes additional overhead. I do not mean to say that a sinusoidal network has no problems, it has several, but by comparison it also demonstrates an extraordinary learning ability that I have not seen matched in other systems.

Simulations:

All programs described in succeeding paragraphs have been implemented on a Symbolics 3670 using the Zetalisp flavors package. Communication between nodes occurs via message passing, and each node is configured to use both a learning rate coefficient and a momentum term. Learning rates affect the amount of error actually propagated back through the network. Momentum terms discourage the learning from getting caught in local function minima by pushing the learning in an established direction. Several other parameters also influence network performance. These include the initial values of the resting biases and

connection biases, both of which establish how far the learning must proceed before an exact solution is discovered. The amount of error for an exact solution also varies. For the most part Rumelhart et. al. require that calculated answers lie within 10% of the expected result, although in some instances they relax this constraint to 50% by assuming that any solution above .5 rounds to 1 and that solutions below .5 round to 0. Within their framework, perfect answers cannot be generated as a logical extension of one term in their delta rule, namely $O_{pj}(1 - O_{pj})$. If the output reaches either a 1 or a 0 this term falls to zero and wipes out the other terms. Realistically, they can set the error level arbitrarily small, however, so this poses no problem. Given an error level of 10%, the sinusoidal network discovered these values for connection biases:

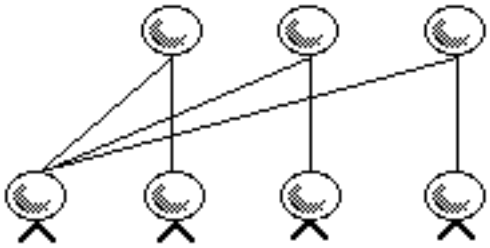


After setting the error to 1e-15 the system arrived at connection biases of 2.19562, - 2.19562, and .785398 which effectively reduce the error to zero. This configuration assumed resting biases given at $\pi/2$ although they equally well could have been set to zero and ignored completely. The most interesting aspect of the net, however, is its learning rate. The optimally configured hidden node network using four nodes discovered a solution in 558 sweeps of the set of input patterns. The optimally configured sinusoidal network, on the other hand, learned the solution in 5 passes over the input data. Yet even more impressive is its learning linearity with respect to the amount of error in the output. Each order of magnitude improvement in the error causes only about 5 additional passes through the data. In short, with the maximum error set to 1e-15 the sinusoidal network learned the answer in only 60 passes. This improves on the hidden node learning algorithm by more than a dozen orders of magnitude using only one tenth the number of iterations. Around 80 passes through the data, the learning algorithm approached the floating point rounding limits of the machine. A table of the sinusoidal network's performance confirms this.

Learn Rate	*Momentum*	*Error*	*nit. Vals*			*Iterations*
2	.1	.1	1	1	1	24
1.5	.1	.1	1	1	1	10
1	.1	.1	1	1	1	11
1	.1	.1	10	20	30	12
1	.1	.1	0	0	0	19
1	.1	.1	π	π	π	5
1	.1	.1	π	2π	3π	8
.7	.1	.1	1	1	1	22
.7	.3	.1	1	1	1	22
.7	.5	.1	1	1	1	22
.7	1	.1	1	1	1	17
.7	1	.01	1	1	1	19
.7	.3	.01	1	1	1	26
.7	.5	.01	1	1	1	27
1	0	.01	1	1	1	15
1	.5	.01	1	1	1	18
1	.2	.01	1	1	1	13
1	.2	.01	π	π	π	8
1	.2	.001	1	1	1	17
1	.2	.001	π	π	π	13
1	.2	.0001	1	1	1	20
1	.2	.0001	π	π	π	16
1	.2	.00001	1	1	1	24
1	.2	.00001	π	π	π	19
1	.2	.000001	1	1	1	28
1	.2	.000001	π	π	π	22
1	.2	1.0e-15	1	1	1	60
1	.2	1.0e-15	π	π	π	55

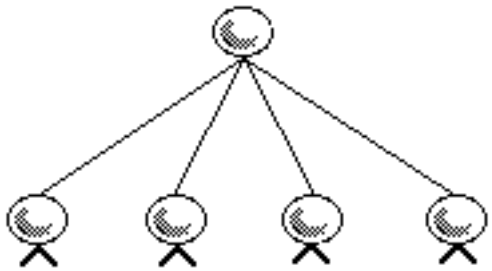
At least two peculiarities stand out in this data as compared with the hidden node system. First, the hidden node system has an optimal learning rate of about .5; it must take smaller steps to reach precise answers. Second, the momentum term seems to have little effect on the number of iterations. The change in momentum from .1 to .3 to .5 with a learning rate of .7 produces no change whatsoever in the number of iterations. Both facts together imply that the solution space for this particular problem has extremely smooth contours and an easily discoverable global minimum. A well-defined minimum such as this portends well for the algorithm in more general applications. In contrast, the occasions where the hidden node system has trouble with local minima occur when multiple hidden nodes in the network do not serve similar functions and are assigned improper responsibilities. The power of hidden nodes also lies in the different internal uses of hidden nodes. There is a tradeoff. As the sinusoidal network possesses great symmetry, it avoids the local minima; however, the network has trouble distinguishing a palindrome because it lacks the additional nodes to represent them. Adding hidden nodes to the sinusoidal formalism is a topic for further study.

Results similar to those for the XOR network obtained for the negation network although the learning savings is still more impressive. Briefly, this network solves for a one's complement to the input. When presented with 0 1 1 1, the system should respond with 1 1 1; if presented with 1 1 1 0, it should produce 0 0 1 as the leftmost input represents a negation bit. The network to accomplish this looks like this:



This network actually achieves an error level of 1e-15 in only 21 passes through the set of sixteen patterns. The comparable hidden node network requires more than 4000 passes through the same data to achieve an error level of .1 . One can best understand such extraordinary performance in terms of a system perfectly tuned for XORing. This negation network reduces to nothing more than three XOR networks connected in parallel.

Because it does not simply reduce to subsets of XOR problems, the parity problem poses a more interesting challenge to a sinusoidal network. The configuration presented here,



shows a four bit parity detection network. If an even number of 1s and 0s appears in the input then a 1 must appear in the output. At an error level of .1, this network discovers the solution in 3 passes through the set of sixteen patterns. At a level of .01 it manages a solution in 39 passes, but at a level of .001 it requires 928 passes. When the network does not decompose into XOR subsystems, the learning rate deteriorates rapidly. Still, a comparable hidden node system plods through 2,825 iterations to reach the .1 error level. Again, most symmetric problems converge more rapidly to a solution in a sinusoidal network. Observing the degradation of a system provided with only partial input provides more interesting insight, however. What happens to a system when it sees data it has not "learned" before? This depends largely on what data it has missed in the past. This net

degrades gracefully in a linear fashion with up to six omissions. In other words, it requires roughly six additional passes through the remaining data for each omission before converging on a solution which will still work on unseen data. Beyond this point, system performance depends on which data are omitted. For example, if it misses the middle eight inputs (binary 4 through 11 out of 0 to 15) it rapidly reaches a solution that works on the known data but is exactly reversed on the foreign data, i.e., in those instances where it should produce a 1, it produces a 0 instead and vice versa. On the other hand, if the eight missing inputs are evenly spaced throughout the data and the error level is set to .0001, it still reaches a proper solution on the unseen data but at an error level of .1 . Any truly intelligent system must posses a version of an extrapolation feature to handle anomalous situations it has never seen. Intelligent programs should reach valid conclusions based on their limited experience and, if they fail, learn from their errors. Sinusoidal networks demonstrate a promising though rudimentary form of this extrapolation.

Context Mechanisms:

Thus far I have neglected the role of a resting bias in node learning and processing. If a resting bias does not change then it plays little role in processing as the connection bias may always change through learning to compensate for it. Further, during simple learning it plays little role because as an additive constant, the partial derivative of the error with respect to a resting bias is negligible. Nevertheless, it has a very significant purpose. If a resting bias changes over time as some function of node stimulation it can assume the role of a local memory. By storing the influences of processed input, a resting bias gives new input the benefit of context. To account for context, I define a resting bias that changes over time as an average of its current bias and total node output minus the initial resting bias. Averaging the current resting bias with total node output assures that new input leaves a context residue behind while averaging with respect to the initial resting bias causes an exponential decay toward the initial value when there is no new input. Thus more recent data provides the best context whereas the absence of new data pushes the system back towards its home state.

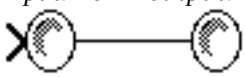
The concept of a resting bias remains completely distinct from the separate concept for a sinusoidal network. Systems of hidden nodes, indeed, any system employing an output function, could incorporate a resting bias. Including such a term generates two problems. Error is not considered until the end of a sequence, requiring that nodes store their intermediate values for use in determining the contribution of each to the overall error. Also, it becomes necessary to rederive the delta rule since the resting bias now changes as a function of the connection bias.

In an effort to capture as much information as possible, I reinstated connection weights and allowed them to change during the learning process. This offers another degree of freedom and increases the chances for a solution. Error in a delta rule modified to include both terms is given by the sum of each of their respective contributions:

$$\partial E_p / \partial O_{pk} \; (\; \partial O_{pk} / \partial ß_k \; + \; \sum_j \partial O_{pk} \; / \; \partial W_{kj} \;)$$

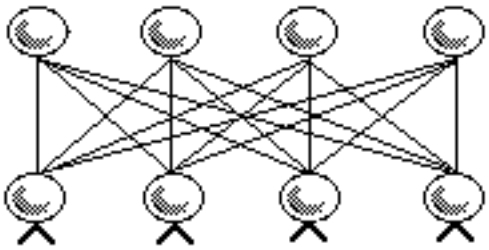
where again E_p is the net error from pattern p, O_{pk} yields the output of k given pattern p, $ß_k$ is that node's connection bias, and W_{kj} is the weight between nodes j and k. Applying the chain rule to this equation is fairly straightforward
although extremely tedious. Suffice it to say that an example runs for the degenerate case of one input node leading into a single output node where the system adjusts both the connection biases and weights to yield:

input: 0 1 *output:* 0



input: 1 1 *output:* 1

The system accepts data sequentially and compares the output to the net result for the entire sequence. I also attempted a larger network of four nodes with only limited success. (Simulation dribble files for both these and all preceding networks accompany this paper) I presented the network



with sequences of up to four inputs although I had no successes with sequences of longer than three.

English language examples serve to illustrate the potential power of such a system. If a network were presented with the two sequences "water well" and "eat well" it would have to disambiguate the final input according to the preceding word. I used vectors of 1s and 0s as the digital analog to these English phrases. Here, the system tried to disambiguate a final input vector consisting entirely of 1s in order that it might produce different output strings of 1s and 0s depending on the preceding strings. How well such a system can perform is determined by the orthogonality of the input vectors and the output

vectors. The greater the difference in *input* vectors, i.e. the more orthogonal they are, the greater is the context the input provides. On the other hand, parallel *output* vectors require little context

because the same final item from the sequence could produce the answer regardless of context. The number of such sequences a network can memorize therefore depends on the degree of context the inputs provide and that the outputs require.

Existing Counterparts:

Proposing a new formalism runs the risk of solving one problem at the expense of creating others. This has, to some extent, happened here. Unfortunately, the network connections and the trigonometric functions do not learn to compute "and" as quickly as they learn to compute "xor." The above examples illustrate how well adapted the system is for computing "xor," however, the input to output mapping does not rise and fall in the same periodic fashion for "and." One consequence of this resulted in my having to reintroduce the concept of a threshold.

Mathematically, one prefers not to resort to the use of thresholds because they introduce discontinuities into what might otherwise be very smooth and functionally elegant equations. Only when output exhibits undesirable behavior do we generally resort to these mathematical exceptions. Here, I experimented with two different types of thresholds. The first type of threshold is fairly common. If the input to a node failed to reach a certain cutoff it was eliminated, meaning it was insufficient to activate the node at all. In short, if the input to a node fell below the threshold I assumed the node output to be a constant, namely the resting bias.

Shortcomings:

As noted earlier, sinusoidal systems do not come without their own shortcomings. An obvious shortcoming is the relative complexity of computing a sine function versus an exponential or multiplication. The sine term costs considerably more. It seems, judgingfrom the magnitude of savings in the learning times, however, that this will not present a problem. A second concern regards the network's inability to handle certain operations such as palindrome detection. It would not significantly compromise the system simply to adopt the pertinent aspects of the hidden node solutions and possibly integrate the advantages of both systems.

All systems utilizing the generalized delta rule potentially suffer from solution spaces rife with local minima. Again judging from the learning rates, this has not been empirically observed as a problem. The networks converge on solutions in extraordinary times, implying that the spaces posses few if any minima traps. The sole possible exception noted so far comes from the context network in which I was unable to generate sequence

recognition beyond a very short limit. One explanation could be the interaction of two solution spaces, i.e., one for connection biases and one for weights. Their solutions may interfere with one another.

Might it also be possible by accident to choose resting biases that make a solution impossible? For the simpler problems the networks did not seem to object to a variety of different combinations of resting biases. But for the most part, I assumed resting biases which made the solution symmetric about some axis. This assumption can often be extended to larger problems.

Advantages:

A sinusoidal system, by virtue of its nonmonotonicity, exhibits a variety of desirable features. Most importantly, sine nodes posses built in thresholds yet they also have the continuous derivatives required by the delta rule. Since a sine function cannot at any time exceed 1 in magnitude, the obvious threshold of a node may be said always to be less than the connection bias. This differs from a traditional view of a threshold in which a node activates only after receiving a predefined quantity of input. This view of a threshold specifies the maximum amplitude of a node regardless of its inputs, the subsequent node thresholds can thus be calculated from there. Or if one prefers, once the input to a sinusoidal node exceeds a certain threshold the output drops back off. This exhibits the nice biological property that beyond a certain level of stimulation the node fails to respond with increased activity; basically, the cell is spent. If this output attenuation becomes a problem the system can eliminate it by dividing the sine term by the sum of the connection biases on the input nodes as the input will never exceed those levels -- the sine function thus never operates outside $-\pi/2$ and $\pi/2$. In other words, a sinusoidal node can also be forced to act more like a traditional node.

Because sine functions cycle from -1 to 1, any node can alternately activate or inhibit subsequent nodes in the network depending on the input it receives. The nodes have become dual-purpose with respect to traditional nodes whose links are static and always either inhibit or activate.

Also unlike the hidden node solutions, there are no theoretical limits on the precision or exactness of a solution. On many occasions, these networks actually produced an exact solution to the problem which can be verified in the following transcripts of system tests. In the paper on hidden units, it was also noted that these functions have a problem with "symmetry breaking." This is a consequence of similar inputs modifying the connection weights equally and so two node weights might never diverge from one another although the solution requires it. This was never a problem in the sinusoidal system.

One further advantage to these networks bears mentioning. In a traditional network, adding new nodes to the system increases the learning time of the system exponentially. As each new node is added, its connection weights with *every* other node also need to be calculated. Under the formalism presented here, however, the learning time of the system is linear with the addition of new nodes. The nodes themselves contain the only variables requiring any calculation. They may have millions of connections yet these are summed and accounted for in the aggregate. Mathematically, a traditional system of two layers of N nodes each computes on N^2 variables whereas this system computes on only 2N.

The results from studying sinusoidal networks (or nonmonotonic networks) under the proposed configuration are very encouraging. Without question the learning rates on suitable problems are unparalleled in other systems. The difference is several orders of magnitude. Certain classes of problems exist, however, which hidden units solve more elegantly. Sinusoidal systems merely offer a powerful and often preferable alternative to monotonic systems. They are more efficient both in terms of speed and numbers of network connections.