OUTPUT FUNCTIONS FOR PROBABILISTIC LOGIC NODES

C. E. Myers

Imperial College of Science, Technology and Medicine, UK

0. ABSTRACT

PLN nets consist of RAM-based nodes which can learn any function of their binary inputs; they require only global error signals during training, and they have been shown to solve problems significantly faster than nets learning by error back-propagation. Output functions for PLNs may be probabilistic, linear or sigmoidal in nature; this paper deals with designing an output function which yields fastest convergence. Experiments with several small problems support the values derived. Choice of an appropriate output function is suggested to be highly problem-dependent, but heuristics for this selection are outlined.

1. THE PLN AND MPLN MODELS

Probabilistic Logic Nodes (PLNs), defined by Aleksander et. al. (1,2), are a class of RAM-based neuron model, designed to combat some of the deficiencies of more ubiquitous weighted-sum-and-threshold approaches; namely that hardware implementation should be straightforward, learning should not be unreasonably slow, and error-correction should require only a global success/failure signal.

Briefly, a PLN is an augmented RAM, in which the I binary inputs to a node form an address into one of the 2^I memory locations, as shown in Figure 1. The output function operates probabilistically on the value stored at this address. During training, an external input pattern is clamped to the input nodes, addressing one location in each; each value accessed is translated into node output and in turn passed as input to other nodes in the net, until external or visible output is produced. If this external output is correct, the value concerned in each node is adjusted so as to increase the probability that its performance will be repeated when next that location is addressed; if the external output is in error, adjustment is made to decrease this probability.

In the simplest case, that of the three-state PLN, every location in every node contains one of three possible values: 0, 1, and "u" – representing "unknown" for the initialised state. When a "u" is accessed, the output function produces a 0 or 1 with equal probability. Stored values are then adjusted from "u" to 0 or 1, as the probability of outputting a 1 is to be encouraged or discouraged, respectively. When an error occurs, as indicated by the external output deviating from the ideal, all currently accessed stored values are reset to "u". Networks of these units are capable of solving hard learning problems (1). A further advantage of the PLN paradigm is that it does not require a specific error signal indicating what the desired output pattern should have been, nor must it calculate errors at each node. A global signal to the effect that error was detected or absent is generated, and all nodes then update on the basis of this general information.

It is possible, however, that the stored values be elements taken from a much larger range than that described above. For example, the values could be selected from the set {0.0, 0.1, 0.2, 0.3, ... 1.0}, representing the probability of outputting a 1 when that value is addressed. Locations in the node might then undergo incremental changes from randomness (0.5) to certainty (0.0 or 1.0). By allowing incremental adjustments, the net learns even more quickly. Myers and Aleksander (3) compared this model to the error back-propagation results given by Rumelhart (4), and found it to be significantly faster and also to generalise well. This learning algorithm also gains the potential for varying the size of the increment in context of how important the trial is. This is however only one possible output strategy for the PLN; the experiments cited in this paper examine the performance of other ranges and interpretations of value.

2. OUTPUT FUNCTIONS

The output function of a node may be described formally as a rule by which the node determines its output, given a certain pattern on its input lines. In the case of the PLN, it is the mechanism whereby stored values are interpreted as affecting the probability that a 1 is output at that node. In equational form, the output function of the three-state PLN is given as:

$$\mathbf{Prob}(\phi = 1) = \xi, \text{ if } \xi \in \{0, 1\}$$
$$= .50, \text{ if } \xi = \text{"}u\text{"} \quad (1)$$

where ξ is the value stored at the location currently accessed and ϕ is the value output. Similarly, for the 11-state PLN described above, we get:

$$\mathbf{Prob}(\phi = 1) = \xi \quad (2)$$

This is a step-function approximating a linear output function. In this case, it is a fairly gross approximation, since there are only eleven steps. A higher number of elements as possible stored values yields a finer approximation, but more bits of RAM are needed in each node to store the 2^I ξ's.

Because of the mutual independence of the ξ's within a single node, the PLN is not restricted to linear functions, and may execute any arbitrary (non-monotonic, non-smooth, non-differentiable, etc.) function. It is true that linear models such as threshold logic (4), ADALINES (5) and Kohonen nets (6) have had considerable success, but there are several reasons why it is desirable to enable the use of non-linearities, particularly the class of semi-linear or "squashing" functions.

As early as 1978, Brodie (7) described cells in *Limulus* eye with a response that was linear about a central range, but which saturated when presented with extreme stimuli: this of course describes a squashing function. Sejnowski (8) reinforces this point in his model of cortex. Biological mimicry is not, however, the only reason to employ sigmoidal output functions: they are essential for signal/noise separation in reverberating nets. Grossberg (9) neatly shows how a sigmoidal output function dissipates noise and quenches units below threshold, while those above threshold are contour-enhanced and may be stored in the reverberating circuit. Thus, non-linearities are a mathematical necessity in complex systems.

The two PLNs described above may be viewed as having maximally different output functions: *the first is very "hard" – i.e., there is no* intermediate space (except "u") between the conditions of consistently outputting a 0 and a 1. The second, conversely, has a very "soft" limiter, where similar values of ξ yield similar probabilities of outputting a 1. Of course there exist an infinite number of intermediate curves between these two extremes. The current experiments examined effect on speed of learning of PLN output functions varying first in ω, the number of elements representing possible stored values, and second in the softness or hardness of the function which interprets and operates on these values.

3. CHOOSING ω - NUMBER OF POSSIBLE STORED VALUES

A PLN net is said to converge when all locations in all nodes either have probabilities of outputting 1 equal to 0 or 1, or else are never addressed. After convergence, then, any PLN net could be replaced by a set of equivalent three-state PLNs without altering performance.

The advantage of a PLN with many possible stored values during training is that after several reinforcements of a RAM location's value in one direction (e.g., toward outputting a 1), it is very hard to erase that knowledge: in fact, it will take an equal number of negative experiences to return it to "u". Thus it has some protection against errors *which may occur in other parts of the net* but which affect it as well because the error signal is global and indiscriminate. It will be forced to edge back towards "u", but only one step, and the probability of outputting a 1 need only change by very little in any one cycle. In a network where $\omega = 3$, in contrast, a single error arising anywhere in the net results in one location in every node being reset to "u", and thus knowledge is erased, regardless of whether any individual node was in fact responsible for the error.

A cost of increased ω, besides requiring more RAM, is the difficulty of returning a location to "u" when this is required. Noisy data or an unfortunate ordering of training examples could push a location's

value very far from "u" in an undesirable direction, and an equal number of error cycles will be required to reset it.

Ideally, ω must be chosen to balance protection against mistaken erasure versus ability to erase when necessary.

3.1. Definitions

For a given feedforward net of N nodes of fan-in I, with one output node T, describe the current state $S = (S_1, S_2, \cdots S_N)$, and for all nodes i, $S_i = (\xi_{i0}, \xi_{i1}, \cdots \xi_{i2^I})$, where $\xi_{ij} \in R$, the ω-element range of possible stored values. Consider one node i, and let ξ_{ij} be the value stored at the location addressed under the current input j. Then the output of the node, ϕ_i, is given by the output function $\Phi(\xi_{ij}) = Prob(\phi_i = 1)$.

When a net is required to learn a function from input pattern to output pattern, there exist $z \geq 0$ solutions such that in that state the net always gives a correct response. For the current purposes, we consider only problems where $z > 0$.

Then, when the net is in state S, there exist one or more "closest" solutions S_c in function space, where the difference function is at a minimum:

$$f = \sum_{i=1}^{N} \sum_{j=0}^{2^I-1} abs(\Phi(\xi_{ij}^{S}) - \Phi(\xi_{ij}^{S_c})) \tag{3}$$

Given one S_c, call the output from each node in that state ϕ_i^*. Then we may define each ξ_{ij} to be "Right" if when addressed,

$$sgn[Prob(\phi_i = 1) - .5] = sgn[Prob(\phi_i^* = 1) - .5] \tag{4}$$

and "Wrong" otherwise. More generally, we may say that a node outputs the Right/Wrong value with probability R_ξ/W_ξ when a given ξ is addressed. Usually, these probabilities can only be measured at the output node, T.

Then the problem reduces to a choice of ω such that there is a high probability of erasing Wrong values even when $R_T > W_T$, and simultaneously a low probability of erasing Right values even when $W_T > R_T$.

3.2. Choosing low ω to allow desired erasure

A RAM location q in node i is set at $\xi \in R$. At the current net state, P patterns address it; for P^R, $\phi_T = \phi_T^*$ while for P^W, $\phi_T \neq \phi_T^*$. Notice that it is possible that $|P^W| > |P^R|$, even if ξ is Right, or for $|P^R| > |P^W|$ when ξ is not Right, depending on activity elsewhere in the net.

In order to return ξ to random, so it might be reset, there must occur y trials such that the number of elements of P^W appearing exceeds the number of elements of P^R by at least $\left\lfloor \frac{\omega}{2} \right\rfloor$, as in the worst case, ξ is $\left\lfloor \frac{\omega}{2} \right\rfloor$ steps away from "u". For simplicity, we consider only the $x < y$ trials during which ξ is actually accessed, and assume the rest of the net is subject only to minor change, so that S_c does not change.

If clamped training inputs are selected randomly, elements of P^R and P^W occur as Bernoulli trials. So the probability that at least m elements of P^W occur in x trials is:

$$Prob(X \geq m) = B(x,m) + B(x,m+1) + \cdots + B(x,x) \tag{5}$$

where $B(u,v) = \binom{u}{v} W^v R^{u-v}$, $W = \frac{|P^W|}{|P|}$, $R = \frac{|P^R|}{|P|}$, and $x \geq m$. But this is not strict enough: it does not suffice that m events occur unless

$$m \geq (x - m) + \left\lfloor \frac{\omega}{2} \right\rfloor, \text{ or } 2m - x \geq \left\lfloor \frac{\omega}{2} \right\rfloor \tag{6}$$

So, the probability that enough elements of $|P^W|$ appear in x trials to reset ξ to randomness is given by:

$$P^1 = Prob(\xi \text{ reset within } x \text{ trials}) = \sum_j \sum_k B(j,k) \tag{7}$$

where j ranges from $\left\lfloor \frac{\omega}{2} \right\rfloor$ to x, k ranges from $\frac{1}{2}\left(\left\lfloor \frac{\omega}{2} \right\rfloor + j\right)$ to j, and $\sum$ summates increasing its index by 2.

3.3. Choosing high ω to avoid mistaken erasure

This time, to avoid ξ being reset to random, there must not occur y trials (of which some x are relevant) such that the number of patterns accessing ξ and generating $\phi_T \neq \phi_T^*$ exceeds those generating $\phi_T = \phi_T^*$ by at least $\left\lfloor \frac{\omega}{2} \right\rfloor$. The probability that this does not occur is:

$$P^2 = Prob(\xi \text{ not reset in } x \text{ trials}) = 1 - P^1 \tag{8}$$

3.4. A compromise

The desired solution maximises both P^1 and P^2: if we consider a sum-of-squares function,

$$(P^1)^2 + (P^2)^2 = P^{1^2} + (1 - P^1)^2 = 1 - 2P^1 + 2(P^1)^2 \tag{9}$$

a maximum occurs at $P^1 = +\frac{1}{2}$, i.e., where $P^1 = P^2 = \frac{1}{2}$. Therefore, the equation to solve is

$$f(\omega) = \sum_j \sum_k B(j,k) \approx \frac{1}{2} \tag{10}$$

Notice that this equation depends on R_ξ, W_ξ, and ω, but not on N, I, or the depth of the net.

Figure 2 shows optimal values of ω, such that (10) is satisfied, as a function of P^W for sample values of x. Ideally, the net should be constructed to minimise ω (and hence the amount of RAM needed) and also x (and hence convergence time). If it is reasonable to assume that P^W deviates around a mean of 0.5, it is seen from Figure 2 that for small x, optimal ω clusters in a range $5 \leq \omega \leq 15$.

Experimental data were gathered to check this assumption and derived ω. Figure 3 shows speed of convergence for small nets of 3 and 7 nodes learning the parity problem at different ω. In the case of the smallest net, $\omega = 11$ led to fastest convergence, while in the larger case a smaller ω performed somewhat better, although for about 75% of the nets, convergence occurred in roughly the same time for all values of ω.

Finally, the nets were set a task involving generalisation. The problem was to recognise a patch of 3 or more 1s in a five-bit input string, including the cases where the patch had moved partially offscreen but at least two of its bits showed. The negative patterns were instances of single patches of 1 or 0 bits, or patches of 2 bits centered – and thus not tips of a larger patch. Figure 4 shows the training set and learning speeds. Nets with $\omega = 11$ learned the distinction fastest, with 76% of sample nets converging in less than 100 training cycles; the $\omega = 21$ nets lagged well behind the others, and in fact 54% failed to converge within 5000 cycles.

After each net had been trained, it was required to respond to all of the $2^5 = 32$ possible inputs. Of the 16 unseen patterns, 7 were examples of positive patterns distorted by one bit; 9 were negative ones with one bit of noise. Of the total 32 patterns, $\omega = 6$ nets averaged 27.42 correct responses; $\omega = 11$ nets averaged 27.45 correct, and $\omega = 21$ averaged 28.30 right. In all cases a reasonable degree of generalisation was achieved. However, the $\omega = 21$ nets showed the most capability to adapt their responses to new stimuli. Because in nearly half the cases the $\omega = 21$ nets had failed to converge, some nodes i in the net had locations such that, when accessed, $0 < Prob(\phi_i = 1) < 1$. Therefore, there was a possibility of outputting different values at these nodes, even given the same address. This flexibility is responsible for the nets' ability to generate a wider range of input in response to new patterns, and hence results in a slightly higher percent correct on the generalisation task.

Overall, empirical data tend to support the theoretical prediction: that $5 \leq \omega \leq 15$, and in specific, $\omega \approx 11$, should lead to fastest convergence.

For the remainder of this paper, ω will be held constant to the value of 11.

4. CHOOSING Φ - THE OUTPUT FUNCTION

The second main concern of this paper is selection of Φ, the output function. A node i addresses location q which stores some value $\xi \in R$. Then its output ϕ_i is determined by:

$$\Phi(\xi) = Prob(\phi_i = 1) \tag{11}$$

Φ may range from relations which approximate linear functions to ones which approximate threshold functions.

4.1. Derivation of Φ

Consider again a feedforward tree, with one output node, T. Define R_ξ as the Right value of ξ with respect to a nearest solution S_c, and define ϕ_i^* as the Right output from node i under the current input. ξ is adjusted toward R_ξ when $\phi_i = R_\xi$ and $\phi_T = \phi_T^*$, and thus repetition of the event is encouraged. Similarly, when $\phi_i \neq R_i$ and $\phi_T \neq \phi_T^*$, the current event is made less likely to reoccur, and the desired one encouraged. If either of the other two events occur ($\phi_i = R_\xi$ and $\phi_T \neq \phi_T^*$, or $\phi_i \neq R_\xi$ and $\phi_T = \phi_T^*$), ξ is trained wrongly – i.e., away

from R_ξ. It is therefore desirable to select Φ such that for a value ξ addressed in node i:

$$Prob(\phi_i = R_\xi \cap \phi_T = \phi_T^*) + Prob(\phi_i \neq R_\xi \cap \phi_T \neq \phi_T^*) >> Prob(\phi_i = R_\xi \cap \phi_T \neq \phi_T^*) + Prob(\phi_i \neq R_\xi \cap \phi_T = \phi_T^*) \quad (12)$$

Assume $R_\xi = 1$ and $\phi_T^* = 1$; the other cases are of course symmetric. Then (12) reduces to:

$$\Phi(\xi)\sum_v Prob(v \text{ accessed}|\phi_i = R_\xi)\Phi(v)) + (1-\Phi(\xi))\sum_v Prob(v \text{ accessed}|\phi_i \neq R_\xi)(1-\Phi(v)) >> \Phi(\xi)\sum_v Prob(v \text{ accessed}|\phi_i = R_\xi)(1-\Phi(v)) + (1-\Phi(\xi))\sum_v Prob(v \text{ accessed}|\phi_i \neq R_\xi)\Phi(v) \quad (13)$$

where v is a value stored in some location in T, and $Prob(v\text{accessed})$ represents the likelihood that the location containing v is addressed given the current output from node i. $Prob(v \text{ accessed}|\phi_i = R_\xi)$ is of course independent of $\Phi(v)$ and $\Phi(\xi)$ This simplifies to:

$$\Phi(\xi)\sum_v Prob(v \text{ accessed}|\phi_i = R_\xi)\left[2\Phi(v)-1\right] >> (1-\Phi(\xi))\sum_v Prob(v \text{ accessed}|\phi_i \neq R_\xi)\left[2\Phi(v)-1\right] \quad (14)$$

A general form for Φ is $(1+e^{-x})^{-1}$. We consider a particular instantiation, $\Phi(x) = (1+e^{\alpha(-2x+1)})^{-1}$ – allowing x to range from 0 to 1 instead of the more usual -1 to +1. The topic of concern is then to select α to maximise inequality (15).

$$\frac{1}{1+e^{\alpha(-2\xi+1)}}\sum_v Prob(v \text{ accessed}|\phi_i = R_\xi)\left[2\Phi(v)-1\right] >> \frac{e^{\alpha(-2\xi+1)}}{1+e^{\alpha(-2\xi+1)}}\sum_v Prob(v \text{ accessed}|\phi_i \neq R_\xi)\left[2\Phi(v)-1\right] \quad (15)$$

or,

$$\frac{\sum_v Prob(v \text{ accessed}|\phi_i = R_\xi)\left[2\Phi(v)-1\right]}{\sum_v Prob(v \text{ accessed}|\phi_i \neq R_\xi)\left[2\Phi(v)-1\right]} >> e^{\alpha(-2\xi+1)} \quad (16)$$

The right side of this equation is a constant for a given value of ξ, and has a minimum as $\xi \to 1$, depending on the assumption $R_\xi = 1$. The left side, however, will change dramatically as the net is trained, since it depends on the probability of accessing different locations in the top node given an output from i. Therefore, the best solution to this equation is to make α as large as possible, to maximise the frequency with which states in the net will satisfy the inequality. The result is an output function, Φ, which should be made to resemble a very steep sigmoidal curve.

The result is intuitively satisfying: it suggests that once a node location is 'committed' to an output, i.e., that it has been reinforced even once away from randomness and toward either 1 or 0, it should output that value consistently. This allows other locations in the net to organise around one another with some confidence that all are behaving as they expect to behave when fully trained.

4.2. Experimental Results

Five limiters were compared in performance: approximating

$$\Phi(x) = \frac{1}{1+e^{\alpha(-2x+1)}} \quad (17)$$

where $\alpha \in \{2.5, 4, 5, 10, 25\}$ and $x \in \{0, 0.1, 0.2, \ldots 1.0\}$. The correspondences between the continuous functions and the Φ approximations are shown in Figure 5.

One hundred nets were trained with each value of a for the two parity problems described in the previous section; in both tasks, there was a clear tendency for nets with hardest limiters to converge fastest and those with softer limiters to converge more slowly. Figure 6 shows these results, and the mean and median time to convergence for each type of net is given in Table 4a.

Set the generalisation task described above, all 5 categories of net tended to learn to recognise the training set in approximately the same time, with 50% of all trials resulting in convergence within 1500 pattern presentations. Out of the 32 patterns in the generalisation set, nets with $\alpha = 2.5$ responded correctly to an average 27.45 patterns; those with $\alpha = 4$ to an average 27.58 patterns, and those with $\alpha = 5$, 10 and 25 to averages of 27.26, 27.32 and 27.20 patterns, respectively. These success rates are all very similar, as expected, since after convergence there should be no difference between nets using any particular limiter in their ability to execute an associative function. As all classes of net converged in similar time, and therefore all had a similar number of trials unconverged at the end of 5000 pattern presentations, no one class of net showed the advantage in generalisation of the unconverged nodes in the $\omega = 21$ nets of the previous section.

These results support the theory that a hard limiter, for example a high value of α in the functions considered here, leads to faster convergence, while having little effect on performance after training is completed.

5. CONCLUSIONS

Several assumptions are implicit in the results presented here, and they warrant restatement. The nets considered are feedforward pyramids, being trained on problems for which a solution exists, via a training schedule which involves a random ordering of training patterns. This is a constrained class of topology and task, but one which is still quite powerful.

Given these assumptions, an MPLN net may be designed which will tend to converge as fast as possible: namely, its nodes contain stored values selected from a range of $5 \leq \omega \leq 15$ elements, and interpreted according to a threshold-like output function. The experiments described to support these claims are small both in terms of the number of nodes and also in terms of size of state space relative to number of solutions available. They are useful, however, since a small number of distinguishable solutions exist and since the parity problems are arguably the "hardest" of the hard learning problems.

It is not the case that the ω and Φ defined here are universally optimal; it is not clear in the first place that speed of convergence is a necessary criterion to judge the "success" of a network – although it is probably the most frequent. There are occasions when a soft limiter, for example, will be desirable despite its slowness. One obvious example involves a state space with abundant and deep local minima, where probabilistic noisy outputs are necessary; in effect, a net using a steep output function forms quick and binding opinions, whereas a net with a more linear output function makes conservative ones, which still allow occasional lapses into the opposite output. This ability would prove important in simulation of an automata existing in a changing environment, where convergence per se is not possible, and where a net might be more successful if some of its nodes, say, output a one 75% of the time, and occasionally output a zero to test the effects in the current environment.

Appropriate choice of parameters is therefore highly dependent on the size, shape and complexity of the problem space, and also causes subtle changes in the way the net organises to solve the problem – particularly in the speed with which nodes commit to a particular output in response to an input pattern. The values of ω and Φ presented here cannot therefore be purported to be optimal under all conditions, merely as especially useful and as good first approximations for later fine-tuning, as necessary.

TABLE 1 - Performance data for experiments on learning 4-bit and 7-bit parity. Mean (with standard deviation) and median times to convergence are shown, along with time for 90% and 100% of nets tested to converge.

3-Bit Parity

α	mean	st.dev.	median	90%	100%
2.5	241	132K	150	500	3000
4	165	21K	150	400	900
5	186	30K	150	450	1000
10	123	15K	70	300	700
25	81	9K	70	200	705

7-Bit Parity

α	mean	st.dev.	median	90%	100%
2.5	6.8K	46K	5.0K	14K	35K
4	8.9K	66K	6.5K	25K	45K
5	7.2K	45K	5.0K	15K	35K
10	4.8K	28K	3.5K	11K	35K
25	3.7K	28K	2.5K	9K	25K

Acknowledgements

I am grateful to Igor Aleksander and the Neural Systems Engineering Group for encouragement and useful discussions. This research was supported by the NSF (USA).

6. REFERENCES

1. Aleksander, I., 1988, in Eckmiller, R and von der Malsburg, C., eds., "Neural Computers," Springer-Verlag, Berlin, 189-197.
2. Kan, W. and Aleksander, I., 1988, in IEEE Proceedings International Conference on Neural Networks, San Diego, 541-548.
3. Myers, C. and Aleksander, I., 1988, in Proceedings First INNS Annual Meeting, Boston.
4. Rumelhart, D., Hinton, G., and Williams, R., 1986, in Rumelhart, D. and McClelland, J., eds., "Parallel Distributed Processing: Explorations in the Microstructure of Cognition", vol. 1, MIT Press, London, 318-362.
5. Widrow, B. and Winter, R., 1988, Computer, 21, 25-39.
6. Kohonen, T., 1984, "Self-Organisation and Associative Memory", Springer-Verlag, New York.
7. Brodie, S., Knight, B. and Ratliff, F., 1978, J. Gen. Psychol., 72, 129-154, 162-166.
8. Sejnowski, T., 1981, in Hinton, G. and Anderson, J., eds, "Parallel Models of Associative Memory", Lawrence Erlbaum, Hillsdale, NJ, 189-212.
9. Grossberg, S., 1973, Studies in Appl. Math., 50, 213-257.

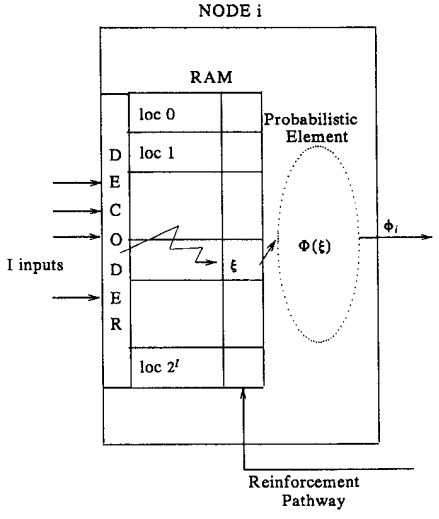
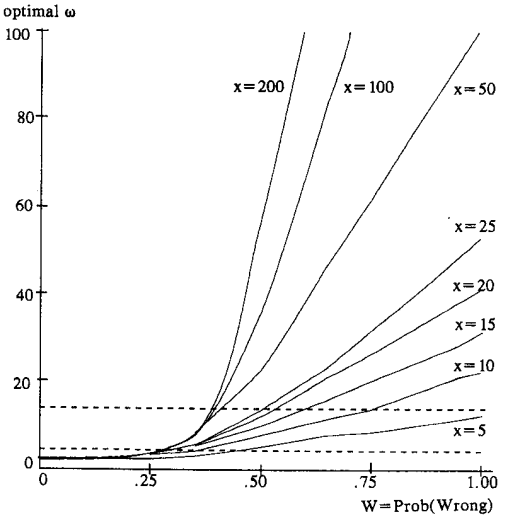


Figure 1 The PLN; I binary inputs address a location in RAM memory; the value accessed, ξ, is passed to the probabilistic output function, Φ, which converts it to binary output, ϕ_i.



Dashed line indicates region of $5 < \omega < 15$

Figure 2 The optimal values of ω, as given by Equation (10), as a function of P^W; plotted for several values of x, the number of training cycles.

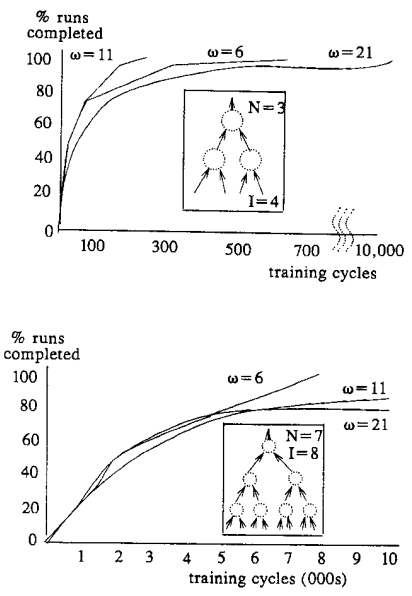


Figure 3 Speed of convergence for two parity problems, for several values of ω. Net topologies are shown in insets; sample size is 100 nets for each ω.

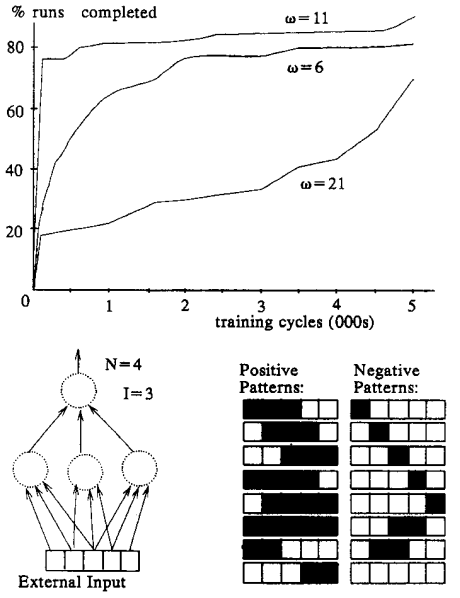


Figure 4 Speed of convergence on the generalisation task for several ω. Net toplology and the 16 training patterns are shown; sample size is 100 nets for each ω.

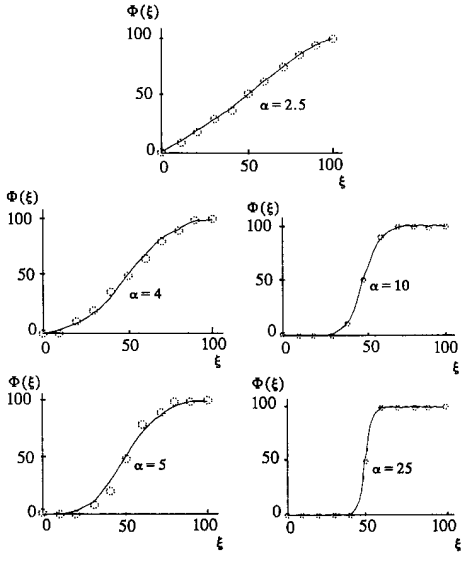


Figure 5 The 5 Φ tested. Solid line is $1/1+e^{\alpha(-2x+1)}$; open circles are actual probabilities used for each value of ξ.

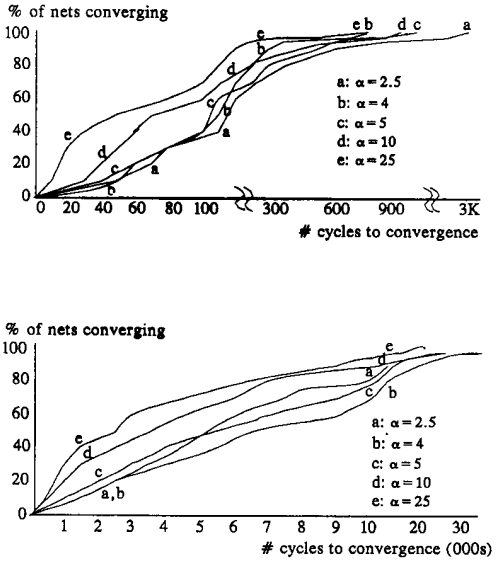


Figure 6 Speed of convergence for two parity problems, for several Φ. Topologies as in Figure 3; sample size is 100 nets for each Φ.