Complex Systems **2** (1988) 39–44

Scaling Relationships in Back-propagation Learning

Gerald Tesauro
Bob Janssens
Center for Complex Systems Research, University of Illinois at Urbana-Champaign
508 South Sixth Street, Champaign, IL 61820, USA

Abstract. We present an empirical study of the required training time for neural networks to learn to compute the parity function using the back-propagation learning algorithm, as a function of the number of inputs. The parity function is a Boolean predicate whose order is equal to the number of inputs. We find that the training time behaves roughly as 4^n, where n is the number of inputs, for values of n between 2 and 8. This is consistent with recent theoretical analyses of similar algorithms. As a part of this study we searched for optimal parameter tunings for each value of n. We suggest that the learning rate should decrease faster than $1/n$, the momentum coefficient should approach 1 exponentially, and the initial random weight scale should remain approximately constant.

One of the most important current issues in the theory of supervised learning in neural networks is the subject of scaling properties [4,13], i.e., how well the network behaves (as measured by, for example, the training time or the maximum generalization performance) as the computational task becomes larger and more complicated. There are many possible ways of measuring the size or complexity of a task; however, there are strong reasons to believe that the most useful and most important indicator is the predicate order k, as defined by Minsky and Papert [6]. The exact technical definition of the predicate order of a Boolean function is somewhat obscure; however, an intuitive way of understanding it is as the minimal number of input units which can provide at least partial information as to the correct output state. While analytic results for scaling dependence on k has not been carried out for back-propagation or Boltzmann machine learning, analysis of related algorithms [1,2,14,15] suggests that the minimum training time, and the minimum number of distinct training patterns, should increase *exponentially* with the order k, probably as $\sim c^k$, where c is some numerical constant, and possibly as bad as $\sim n^k$, where n is the number of input units.

In this brief technical note, we report an empirical measurement of the scaling of training time with predicate order for the back-propagation learning paradigm. The task chosen for training the networks was the parity

function, i.e. the function which returns 1 if an odd number of inputs units are on, and 0 if an even number of input units are on. We chose the parity function because of its simple definition and symmetry properties, because an analytic solution for the network weights is known [9], and because n-bit parity is known to have predicate order $k = n$. This can be seen from the intuitive definition of predicate order. For n-bit parity, no subset of input units of size less than n provides any information as to the correct answer; this must be determined by observing all n input values. Thus, the parity function has the maximum possible predicate order, and in that sense should be representative of the hardest possible functions to learn using standard back-propagation learning.

Our empirical study was carried out as follows. For each value of n examined, we set up a back-propagation network with n input units, $2n$ hidden units, and 1 output unit, with full connectivity from inputs to hiddens and from hiddens to output. We chose to use $2n$ hidden units, because the minimum required number of hidden units can be shown to be n [9], and with more than the minimal number the problem of trapping in local minima should be reduced [8]. The initial weights were set to random values on a scale r which was varied as a part of the study. The 2^n training patterns were then cycled through in order, and the weights were adjusted after each pattern. The learning rate and momentum parameters ϵ and α, as defined in [12], were also varied in this study. The training continued until the network performed correctly on each of the 2^n possible patterns, with correct performance being defined as coming within a margin of 0.1 of the correct answer. If the network had not reached perfect performance by a certain cutoff time, the training run was abandoned, and the network was reported to be stuck in a local minimum.

The question of parameter tuning is highly non-trivial in this study. When comparing results for two different networks of size n and n', it is not at all clear *a priori* what setting of parameter values constitutes a fair comparison. We decided to search for, at each value of n, the values of ϵ, α and r which gave optimal average training time. This necessitated trying empirically several different combinations of parameter settings, but produced added interesting results as to how the optimal parameter values scale with increasing n.

Early in the study, we found that the learning of the parity function is an extremely noisy stochastic process, with wide variations in the training time from run to run over many orders of magnitude. In order to achieve good statistics on the average training time, it was necessary to do 10,000 learning runs for each data point (n, ϵ, α, r). This produced average training times with at least two significant figures. There is an additional problem in that for the learning runs which become stuck in local minima, the training time is infinite, and thus a direct averaging of training times will not work. We defined the average training time to be the inverse of the average training rate, where individual training rates were defined as the inverse of individual training times. With this definition, there is no problem in computing the average, since the training rate for a run stuck in a local minimum is zero.

n	ϵ_{opt}	α_{opt}	r_{opt}	T_{opt}	$T_{opt}/2^n$	$T_{opt}/4^n$
2	30 ± 2	$.94 \pm .01$	$2.5 \pm .2$	95 ± 2	24	5.9
3	20 ± 2	$.96 \pm .01$	$2.5 \pm .5$	265 ± 5	33	4.1
4	12 ± 2	$.97 \pm .02$	$2.8 \pm .5$	1200 ± 60	75	4.5
5	7 ± 1	$.98 \pm .01$	$2.5 \pm .5$	4100 ± 300	130	4.0
6	3 ± 1	$.985 \pm .01$	[2.5]	20000 ± 2500	310	4.9
7	[2]	[.99]	[2.5]	[100000]	800	6.1
8	[1.5]	[.99]	[2.0]	[500000]	2000	7.6

Table 1: Measured optimal averaging training times and optimal parameter settings for n-bit parity as a function of number of inputs n. The optimized learning parameters were: learning rate ϵ, momentum constant α, and initial random weight scale r. For $2 \leq n \leq 5$, 10000 runs at each combination of parameter settings were used to compute the average. For $6 \leq n \leq 8$, only a few hundred runs were made. Quantities in brackets were not optimized.

Our results are presented in table 1. Note that the average training time increases extremely rapidly with increasing n. Since there is a further increase in simulation time proportional to the number of weights in the network ($\sim n^2$), it became extremely difficult to achieve well-optimized, accurate training times beyond $n = 5$. (We estimate that $\sim 10^{14}$ logic operations were used in the entire study, taking several weeks of CPU time on a Sun workstation.) For the final two values of n, no attempt was made to find the optimal parameter values, and the reported optimal training times are probably only accurate to within 50%. Also note that when the average training times are divided by 2^n (this gives the required number of repititions of each training pattern), the resulting figures are still definitely increasing with increasing n; thus the training time increases faster than 2^n. Dividing by 4^n, however, produces figures which are roughly constant. This provides suggestive evidence (which is by no means conclusive) that the training time increases roughly as $\sim 4^n$. This would be consistent with analysis of related algorithms. Volper and Hampson [1,2] discuss several related learning algorithms which utilize a mechanism for adding new hidden nodes to the network, and which are capable of learning arbitrary Boolean functions. For hard problems such as parity, these algorithms generally require training times $\sim 2^n$, with $\sim 2^n$ hidden nodes; thus training times $\sim 4^n$ with a fixed polynomial number of hidden nodes are reasonable.

Regarding the issue of optimal parameter values, we discuss each in turn. The optimal value of ϵ appears to fall off somewhat faster than $1/n$, possibly as $n^{-3/2}$ or n^{-2}. Hinton [3] suggests on the basis of fan-in arguments that the learning rate should decrease as $1/n$. Our results support the notion that the learning rate should decrease with increasing fan-in, although the exact rate of decrease is undetermined.

The optimal momentum coefficient appears to approach 1 as n increases. We cannot be sure whether the rate of approach is exponential or polynomial;

		ϵ							
		16	20	24	28	32	36	40	44
α	.98			152.0 (.0219)	129.7 (.0235)			124.4 (.0389)	
	.96	139.8 (.0122)		112.2 (.0096)	105.8 (.0136)	101.6 (.0179)	99.5 (.0287)	99.6 (.0482)	102.6 (.0698)
	.94		108.3 (.0105)	99.2 (.0117)	95.0 (.0255)	95.0 (.0425)	98.7 (.0759)	106.1 (.1291)	
	.92	118.0 (.0106)	102.0 (.0097)	95.1 (.0194)	95.9 (.0509)	101.9 (.1047)	111.9 (.1733)		
	.90	116.6 (.0109)	101.2 (.0142)	97.6 (.0418)	102.7 (.0988)				
	.88	116.6 (.0080)	103.3 (.0205)	104.0 (.0672)					

Table 2: Average training times, and in parentheses, fraction of runs stuck in local minima, for 2-bit parity (XOR) as a function of learning rate ϵ and momentum coefficient α. In each, the initial random weight scale was $r = 2.5$.

however, an expontential approach would be consistent with the following simple argument. The quantity $(1 - \alpha)$ is essentially a decay term which indicates how long the weight change due to a particular pattern persists. In other words, the use of a momentum term is in some sense equivalent to computing a weight update by averaging over a number of patterns. If the decay rate decreases exponentially, say as 2^{-n}, this would indicate that the optimal number of patterns to average over increases as 2^n, or that the fraction of patterns in comparison to all possible patterns remains constant.

Finally, regarding the optimal scale of the initial random weights, we were surprised to find that this parameter remained approximately constant as n increased. We would have expected some sort of decrease in the random weight scale. For example, one could argue that the average level of activation of a hidden unit due to n random weights from the input units would be of order $n^{1/2}$; thus the weight scale should decrease as $n^{-1/2}$ to keep the input constant. However, this does not appear to be the case empirically.

It is also interesting to note what happens as the parameters are varied from their optimal values. For all three parameters we find as the parameter is increased, the learning goes faster for runs which do not get stuck in local minima. However, once the parameter goes beyond the optimal value, there is a rapid increase in the fraction of learning runs which do become stuck. Thus the parameter values we report are essentially the largest possible values at which one can operate without getting stuck in local minima an unacceptable percentage of the time. Table 2 provides an illustration of how the average training time and the fraction of runs stuck in local minima behave as the parameters ϵ and α are varied, for the case $n = 2$.

The apparent scaling behaviors we have found for the training times and

parameter values are somewhat counter-intuitive, and motivate the search for an analytic explanation. Also we point out that the requirement that the network learn the parity function perfectly is a severe requirement, which may have contributed to the extremely poor scaling behavior. It would be interesting to measure the training time needed to reach some fixed level of performance less than 100%, although we conjecture that such training times would still increase exponentially.

In conclusion, while it is difficult to extrapolate to the large-n limit on the basis of values of n between 2 and 8, we do have suggestive evidence that the required training time for the back-propagation algorithm increases exponentially with the predicate order of the computation being trained. This would indicate that earlier enthusiatic projections of the use of back-propagation to learn arbitrarily complicated functions were overly optimistic. The learning of such functions, while no longer impossible as with perceptron networks, still remains effectively intractable. Thus back-propagation, and probably all known supervised learning algorithms for connectionist networks, should be used only for learning of low-order computations. For high-order problems such as connectedness and parity, alternative procedures should be investigated.

Acknowledgements

We thank Terry Sejnowski for providing the back-propagation simulator code and for helpful discussions. This work was partially supported by the National Center for Supercomputing Applications.

References

[1] S. E. Hampson and D. J. Volper, "Linear function neurons: structure and training," *Biological Cybernetics*, (1986) 203–217.

[2] S. E. Hampson and D. J. Volper, "Disjunctive models of boolean category learning," *Biological Cybernetics*, (1987) 121–137.

[3] D. C. Plaut, S. J. Nowlan and G. E. Hinton, "Experiments on learning by back propagation," Carnegie-Mellon University, Department of Computer Science Technical Report (1986) CMU-CS-86-126.

[4] G. E. Hinton, "Connectionist learning procedures," Carnegie-Mellon University, Department of Computer Science Technical Report (1987) CMU-CS-87-115.

[5] Y. Le Cun, "A learning procedure for asymmetric network," *Proceedings of Cognitiva (Paris)*, **85** (1985) 599–604.

[6] M. Minsky and S. Papert, *Perceptrons*, (MIT Press, Cambridge, MA, 1969).

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing:*

Explorations in the Microstructure of Cognition. Vol. 1: Foundations, D. E. Rumelhart and J. L. McClelland eds., (MIT Press, 1986) 318–362.

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, **323** (1986) 533–536.

[9] D. E. Rumelhart and J. L. McClelland (eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations*, (MIT Press, 1986).

[10] D. E. Rumelhart and J. L. McClelland (eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 2: Psychological and Biological Models*, (MIT Press, 1986).

[11] T. J. Sejnowski, P. K. Kienker and G. E. Hinton, "Learning symmetry groups with hidden units: beyond the perceptron," in *Evolution, Games and Learning: Models for Adaptation in Machines and Nature*, D. Farmer et al. eds., (North-Holland, 1986) 260–275.

[12] T. J. Sejnowski and C. R. Rosenberg, "Parallel Networks that Learn to Pronounce English Text," *Complex Systems*, **1** (1987) 145–168.

[13] G. Tesauro, "Scaling relationships in back-propagation learning: dependence on training set size," *Complex Systems*, **1** (1987) 367–372.

[14] L. G. Valiant, "A theory of the learnable," *Communications of the ACM*, **27:11** (1984) 1134–1142.

[15] L. G. Valiant, "Learning disjunctions of conjunctions," *Proceedings of the International Joint Conference on Artificial Intelligence*, (1985) 560–566.