Predicting the Upper Bound Values of the Numbers of Hidden Units and Training Cycles for Backpropagation Based Networks

WING KAI, LEUNG
School of Computing
Faculty of Computing, Information and English
University of Central England
Birmingham B42 2SU
UNITED KINGDOM

Abstract: - Backpropagation and its variants have been studied by a number of researchers. However, most of the researchers have experienced difficulties in deciding the appropriate numbers of hidden units and training cycles to solve a specific application. In most cases, these quantities are determined empirically. In addition, no detailed study has been made in generalising the empirical results for each type of application problems. With the generalised results, it is possible that the upper bound values of the numbers of hidden units and training cycles required to solve a specific type of application can be predicted mathematically. This paper attempts to summarise and generalise all the theoretical and experimental results obtained in the previous studies by Leung et al [1,2,3,4,5,6,7] with a view to devising a mathematical approach in predicting the numbers of hidden units and training cycles required to solve a variety types of application problems.

Keywords: - Neural Networks, Backpropagation, Hidden Units, Training Cycles, Neural Metrics, Algorithmic Complexity.

1. Introduction

Since their invention in the mid 1940's, Artificial Neural Networks (ANNs) have been developed and applied in a number of areas including speech and image recognition. The literature in ANNs is scattered over a vast number of publications spanning various unrelated domains. The diversity of these sources makes it difficult for inexperienced neural network users to learn about ANNs and for researchers to keep pace with current developments. The former users, overwhelmed by the sheer volume of recent papers, are unfamiliar with ANN terminology and certain commonly used concepts [8]. Consequently, it is difficult to get a complete picture of ANNs and to combine and apply results from diverse sources to practical problems. Lawrences [9] reported that 85% of neural network researchers produce their own software, much of which is either unavailable to the rest of the research community or is not modifiable for similar research.

In addition, difficulties in implementing such systems in a well specified and computationally measurable manner have restrained research in this area for years. However, the advancements in technology such as the exponential increase in computer hardware power, has resurrected interests in this field in the last decade. One major discovery is the backpropagation of errors training algorithm (BPA) [10] which was applied to Multi Layer Perceptrons (MLP's).

Due to the lack of applicable measurements, no comprehensive analysis has been carried out on the quality characteristics (e.g. efficiency and algorithmic complexity) of any backpropagation based network system simulated on a conventional computer. This is also true for variants of MLP. Hence, users of such systems with little or no specialist expertise in this area do not generally know how efficient the MLP performs nor how complicated the training process is for solving a specific application problem. In addition, the construction of the actual training programs using a programming language applicable to ANNs has rarely been presented. Therefore, it is difficult to acquire and compare implementation details based on the published work of each researcher. Furthermore, researchers sometimes use minor variations on the published algorithms without making it clear what the variations are and why they were necessary.

The main difficulty experienced by most researchers and users is the decision of choosing the appropriate numbers of hidden units and training cycles to solve a specific application. Most studies do not attempt to generalise the results for each type of problem, e.g. the average algorithmic complexity of the N-bit encoding problem and the number of hidden units and training cycles required to solve the problem. By knowing the generalised algorithmic complexity of a specific type of application problem, it is possible that the upper bound values of the numbers of hidden units and training cycles required to solve the same type of problem can be predicted mathematically.

This study is based on the work conducted by Leung et al [1] who proposed and defined the set of Neural Metrics to measure the quality characteristics (e.g. efficiency and complexity) of neural networks. Firstly, all neural metric functions as defined in [1,3,4,7] are generalised so that a single generic set of neural metric functions can be used to compute the algorithmic complexity of a specific problem whether it is being solved by the standard BPA or its variants. Secondly, the algorithmic complexity of each type of problem addressed in [4,6] is generalised so that a prediction on the upper bound values of some of the neural metrics (e.g. the numbers of hidden units and training cycles which would normally need to be determined empirically) can be made on the same type of problem. Finally, the operational requirements in terms of CPU time required to solve each type of problem are also summarised.

2. Generic Neural Metric Functions

In [1,3,4,7], several neural metric functions have been defined in the analysis of the backpropagation based training algorithms. These are the proposed computed metrics, ACT, ADD, MUL and TOT which show the number of activation calls, additions, multiplications and total operations respectively required by the network system to solve the given problem successfully. They can be calculated (using formulae 1 to 6 below) to provide quantitative measures of the efficiency of the network system and the algorithmic complexity of a specific benchmark problem when being solved by backpropagation.

There are essentially two distinct types of neural metric functions, one for continuous and the other for periodic weight update. Depending on the type of optimisation techniques being used during the training process, the definition of a metric function may vary further. By generalising the results obtained in [1,3,4,6,7], a set of the generic neural metric functions for continuous weight update can be defined as

$$ADD(M,m,n^{[1]},n^{[2]},\ldots,n^{[m]}) = M(\sum_{s=2}^{m-1} n^{[s]}(k_1 n^{[s+1]} + k_2 n^{[s-1]} - 1) + k_2 n^{[m]}(k_3 n^{[m-1]} + 1)) \quad (1)$$

$$MUL(M,m,n^{[1]},n^{[2]},\ldots,n^{[m]}) = M(\sum_{s=2}^{m-1} n^{[s]}(k_4 n^{[s+1]} + k_5 n^{[s-1]} + k_6) + k_7 n^{[m]}(k_8 n^{[m-1]} + k_9)) \quad (2)$$

$$ACT(M,m,n^{[1]},n^{[2]},\ldots,n^{[m]}) = M\sum_{s=2}^{m} n^{[s]} \quad (3)$$

$$TOT(M,m,n^{[1]},n^{[2]},\ldots,n^{[m]}) = M(k_{10}\sum_{s=2}^{m-1} n^{[s]}(k_{11} n^{[s+1]} + k_{12} n^{[s-1]} + k_{13}) + n^{[m]}(k_{14} n^{[m-1]} + k_{15})) \quad (4)$$

and for periodic weight update as

$$ADD(P,M,m,n^{[1]},n^{[2]},\ldots,n^{[m]}) =$$
$$PM(\sum_{s=2}^{m-1} n^{[s]}(k_{16} n^{[s+1]} + k_{17} n^{[s-1]} + k_{18}) + k_{19} n^{[m]}(n^{[m-1]} + k_{20})) + M\sum_{s=2}^{m} n^{[s]}(n^{[s-1]} + k_{21}) \quad (5)$$

$$MUL(P,M,m,n^{[1]},n^{[2]},\ldots,n^{[m]}) =$$
$$PM(\sum_{s=2}^{m-1} n^{[s]}(k_{22} n^{[s+1]} + k_{23} n^{[s-1]} + k_{24}) + k_{25} n^{[m]}(k_{26} n^{[m-1]} + k_{27})) \quad (6)$$

$$ACT(P,M,m,n^{[1]},n^{[2]},\ldots,n^{[m]}) = PM\sum_{s=2}^{m} n^{[s]} \quad (7)$$

$$TOT(P,M,m,n^{[1]},n^{[2]},\ldots,n^{[m]}) = PM\left(\sum_{s=2}^{m-1} n^{[s]}(k_{28}n^{[s+1]} + k_{29}n^{[s-1]} + k_{30}) + n^{[m]}(k_{31}n^{[m-1]} + k_{32})\right) + M\sum_{s=2}^{m} n^{[s]}(n^{[s-1]} + k_{33}) \quad (8)$$

Constant	Standard BPA	BPA + Momentum	BPSA
k_0	1	1	2
k_1	2	3	2
k_2	1	1	2
k_3	2	3	1
k_4	1	1	3
k_5	3	4	3
k_6	2	2	5
k_7	1	2	1
k_8	3	2	3
k_9	2	1	4
k_{10}	1	1	5
k_{11}	2	2	1
k_{12}	5	7	1
k_{13}	2	2	1
k_{14}	5	7	5
k_{15}	4	4	7
k_{16}	1	1	2
k_{17}	1	2	1
k_{18}	0	0	-1
k_{19}	1	2	1
k_{20}	2	1	2
k_{21}	0	0	1
k_{22}	1	1	3
k_{23}	3	4	3
k_{24}	2	2	5
k_{25}	1	2	1
k_{26}	3	2	3
k_{27}	1	1	4
k_{28}	2	2	5
k_{29}	4	6	4
k_{30}	3	3	5
k_{31}	4	6	4
k_{32}	5	5	7
k_{33}	0	0	1

Table 1 Constant Values in Generic Neural Metric Functions

where k_n (n = 0, 1, 2, ..., 33) are constants whose values depend on the type of optimisation being used in the training process, $n^{[s]}$ is the number of units in layer s, M is the number of training cycles and P is the number of input patterns. Table 1 shows the values of constants k_n for the cases of standard BPA (where no optimisation techniques are used), BPA with the momentum term (where the momentum term

is used to optimise the training process), and BPSA (where optimisation is made through the gradient descent of the sigmoidal steepness variable [2,5]).

3. Predicting the Numbers of Hidden Units and Training Cycles

It has been suggested and shown in [1,3,7] that most problems being solved by a backpropagation based algorithm have the average algorithmic complexity function $O(N^k)$ where N is the number of connection weights in the network and k is an integer constant $\in$ (3,5). This means each such problem has a polynomial-bound solution and thus belongs to the class of feasible problems. The average algorithmic complexity for each type of problem addressed in this study is generalised and shown in Table 2. It can be seen that each type of problem has the same order of algorithmic complexity. For example, the Encoding problem is $O(N^3)$ whether it is 4-bit, 8-bit or 10-bit. Suggested values of N obtained through this study for each problem are also shown in Table 2. The value of N is dependent on the number of units on each layer of the network, i.e.

$$N = n^{[2]} * (n^{[1]} + n^{[3]}) \qquad (9)$$

where $n^{[1]}$, $n^{[2]}$ and $n^{[3]}$ are the number of units on the input, hidden and output layers respectively. The values for $n^{[1]}$ and $n^{[3]}$ are normally given from the problem but that for $n^{[2]}$ needs to be determined empirically. The results obtained in [4,6] show that

$$n^{[2]} \subseteq P \qquad (10)$$

for P input patterns used in training the network (Table 2). Since P is a known value, the order of magnitude of $n^{[2]}$ can be predicted by formula (10).

Once $n^{[2]}$ is determined, the value of N can be obtained from formula (9). As defined in [1], the algorithmic complexity of a problem is the product of P, M and N, i.e.

$$PMN \subseteq O(N^k) \qquad (11)$$

where the number of training cycles M required to train the network is another quantity that needs to be determined experimentally. However since P is given, N can be obtained from formula (9), and k is constant for the same type of problem, the value of M can be predicted by formula (11).

For example, to predict the upper bounds of the numbers of hidden units and training cycles required in the training of the 6-bit Encoding problem (which was not covered in the Table 2), the following calculations are taken

1. the values of P, $n^{[1]}$ and $n^{[3]}$ are all given as 6.
2. using formula (10), the upper bound value of $n^{[2]}$ is therefore 6.
3. the upper bound value of $N = 6*(6 + 6) = 72$ as from formula (9).
4. the upper bound value of $M = N^k / PN = N^{k-1} / P = 72^{3-1} / 6 = 72^2 / 6 = 864$ as from formula (11) where $k = 3$ for Encoding type of problem (Table 2).

This means a maximum of 6 hidden units and 864 training cycles are sufficient to solve the 6-bit Encoding problem.

Table 2 can serve as a general reference for the average algorithmic complexity involved in solving a particular type of problem. It also provides a categorisation of the average algorithmic complexities for all types of problems addressed in this study. For instance, both the Parity and Binary Addition problems belong to the same category since they have the same order of magnitude of algorithmic complexity. The same is true for the Symmetry and Diabetes problems. In addition, the value of N in the table also indicates the size of the network required for each problem. For example, the size of the network required for the 2-Spirals problem (which has $N \geq 270$ weights) is much larger than that required for the Symmetry problem (which has $N = 20$ weights) even though they have the same order of magnitude of algorithmic complexity.

4. Empirical Results on Application Problems

In addition to evaluating the values of neural metrics for each application problem, this study also evaluated the operational requirement in terms of CPU time taken for each problem. Since the amount of CPU time required to solve a specific problem is machine dependent (e.g. a Pentium II PC with 120 MHz and 16 MB RAM was used in this research), it was not included in [1,3] as one of the neural metrics.

Type of Problem	**Average Algorithmic Complexity**	**Number of Weights *N***	**Number of Input Patterns *P***	**Number of Hidden Units $n^{[2]}$**
Encoding	$O(N^3)$	24 for 4-bit Encoding 80 for 8-bit Encoding 120 for 10-bit Encoding	4 8 10	3 5 6
Parity	$O(N^4)$	15 for 2-bit Parity 28 for 3-bit Parity 50 for 4-bit Parity	4 8 16	5 7 10
Binary Addition	$O(N^4)$	42 for 2-bit Binary Addition	16	6
Symmetry	$O(N^5)$	20 for 4-bit Symmetry	16	4
2-Spirals	$O(N^5)$	greater than 270	194	90
Proben1 - Glass	$O(N^4)$	150	107	10
Proben1 - Cancer	$O(N^4)$	55	350	5
Proben1 - Diabetes	$O(N^5)$	90	384	9

Table 2 Average Algorithmic Complexities of Problems

Table 3 shows the CPU requirement for each type of problems. It can be seen that the value is directly proportional to the number of training cycles required to train the network for each type of problems. Table 3 also shows the values of total squared errors where convergence takes place. For problems (such as those collected in Proben1) with large training data sets, three types of squared errors were measured to evaluate the generalisation performance of the network. These are Training, Validation and Test errors. In general, the Test error is higher than the Training error as the training process takes place. The Validation error provides the actual error measurement for the network.

5. Conclusion

The objective of this study, which attempts to predict the upper bound values of the numbers of hidden units and training cycles required to solve a variety types of application problems, has been achieved. By obtaining such upper bound values, users do not have to spend considerable time and effort to determined empirically the numbers of hidden units and training cycles for the same type of problem. It is also believed that the summarised and generalised results presented in this study may provide users with enhanced knowledge in the average algorithmic complexity and run time requirement (e.g. CPU time) for each type of application problem addressed.

References:

[1] LEUNG W.K. and WINFIELD M., 2000, A Complexity Analysis of the Backpropagation Algorithm, Proceedings of the WSES 2000 International Conference on Applied and Theoretical Mathematics, pp. 2441-6, Athens, Greece, December.

[2] LEUNG W.K. and WINFIELD M., 2000, Implementing Backpropagation with Momentum, Periodic Weight Update and Gradient Descent on Steepness, Proceedings of the WSES 2000 International Conference on Applied and Theoretical Mathematics, pp. 2431-6, Athens, Greece, December.

[3] LEUNG W.K. and SIMPSON R., 2000, Neural Metrics - Software Metrics in Artificial Neural Networks, Proceedings of the KES 2000 International Conference of the Knowledge Base Engineering Systems, Vol 1, pp. 209-12, Brighton, UK, August.

[4] LEUNG W.K., 2001, An Empirical Study of Software Metrics in Artificial Neural Networks, Proceedings of the 5th IEEE/WSES 2001 International Conference on Circuits, Systems, Communications and Computers, pp. 4541-6, Crete, Greece, July.

[5] LEUNG W.K., 2001, The Performance of Backpropagation Networks which use Gradient Descent on Sigmoidal Steepness, Proceedings of the 5th IEEE/WSES 2001 International Conference on Circuits, Systems, Communications and Computers, pp. 4561-6, Crete, Greece, July.

[6] LEUNG W.K., 2001, Solving Application Problems involving Large Real Type Data Sets by Single Layered Backpropagation Networks, Proceedings of the 5th IEEE/WSES 2001 International Conference on Circuits, Systems, Communications and Computers, pp. 4551-6, Crete, Greece, July.

[7] LEUNG W.K., 2001, On the Complexity of Backpropagation with Momentum and Gradient Descent on Steepness, Proceedings of the WSES 2001 International Conference on Neural Network and Applications, pp. 4811-6, Tenerife, Spain, February.

[8] MEHRA P., and WAH B.W. 1992, Artificial Neural Networks: Concepts and Theory, IEEE Computer Society Press.

[9] LAWRENCES J. 1996, Correctness, Efficiency, Extendibility and Maintainability in Neural Network Simulation, Available From :www.neci.nj.nec.com/homepages/lawrence.

[10] RUMELHART D.E., HINTON G.E., and WILLIAMS R.J. 1986, Learning Representations by BackPropagating Errors, Nature Vol. 323, pp. 533-536, October.

Problem	**Number of**	**Train Cys.**	**Total Sq.**	**Errors**	**CPU Time**	**(Second)**
	Mean	Confidence	Mean	Confidence	Mean	Confidence
4-bit Encoding	15 ± 3	15 ± 2	0.10 ± 0.02	0.10 ± 0.01	6.39 ± 0.24	6.39 ± 0.09
8-bit Encoding	70 ± 7	70 ± 3	0.04 ± 0.02	0.04 ± 0.01	8.81 ± 1.11	8.81 ± 0.40
10-bit Encoding	120 ± 7	120 ± 3	0.05 ± 0.06	0.05 ± 0.03	11.74 ± 0.62	11.74 ± 0.23
2-bit Parity	60 ± 8	60 ± 3	0.15 ± 0.03	0.15 ± 0.02	7.41 ± 0.82	7.41 ± 0.30
3-bit Parity	360 ± 11	360 ± 4	0.012 ± 0.004	0.012 ± 0.002	52.10 ± 4.60	52.10 ± 1.65
4-bit Parity	1090 ± 14	1090 ± 6	0.013 ± 0.004	0.013 ± 0.002	150.87 ± 7.55	150.87 ± 2.71
Binary Addition	500 ± 13	500 ± 5	0.015 ± 0.003	0.015 ± 0.002	149.90 ± 16.07	149.90 ± 5.76
Symmetry	170 ± 7	170 ± 3	0.154 ± 0.009	0.154 ± 0.004	77.50 ± 5.90	77.50 ± 2.12
Proben1 - Glass	1000 ± 48	1000 ± 18			171.48 ± 9.10	171.48 ± 3.26
Training Errors			0.069 ± 0.006	0.069 ± 0.003		
Validation Errors			0.073 ± 0.008	0.073 ± 0.003		
Test Errors			0.087 ± 0.006	0.087 ± 0.003		
Proben1 - Cancer	50 ± 5	50 ± 2			60.17 ± 2.36	60.17 ± 0.85
Training Errors			0.019 ± 0.004	0.069 ± 0.002		
Validation Errors			0.023 ± 0.005	0.023 ± 0.002		
Test Errors			0.031 ± 0.006	0.031 ± 0.003		
Proben1 - Diabetes	2000 ± 13	2000 ± 5			249.38 ± 9.28	249.38 ± 3.33
Training Errors			0.022 ± 0.003	0.022 ± 0.002		
Validation Errors			0.024 ± 0.004	0.024 ± 0.002		
Test Errors			0.031 ± 0.004	0.031 ± 0.002		

Table 3 Empirical Simulation Results