Connectivity and Performance Tradeoffs in the Cascade Correlation Learning Architecture*

D. S. Phatak
Electrical Engineering Department
State University of New York, Binghamton, NY 13902–6000

I. Koren
Department of Electrical and Computer Engineering
University of Massachusetts, Amherst, MA 01003

(Technical Report TR–92–CSE–27, ECE Dept., UMASS, Amherst
Last revised: October 1994)

ABSTRACT

The Cascade Correlationis a very flexible, efficient and fast algorithm for supervised learning. It incrementally builds the network by adding hidden units one at a time, until the desired input/output mapping is achieved. It connects all the previously installed units to the new unit being added. Consequently, each new unit in effect adds a new layer and the fan–in of the hidden and output units keeps on increasing as more units get added. The resulting structure could be hard to implement in VLSI, because the connections are irregular and the fan-in is unbounded. Moreover, the depth or the propagation delay through the resulting network is directly proportional to the number of units and can be excessive.

We have modified the algorithm to generate networks with restricted fan-in and small depth (propagation delay) by controlling the connectivity. Our results reveal that there is a tradeoff between connectivity and other performance attributes like depth, total number of independent parameters, learning time, etc. When the number of inputs or outputs is small relative to the size of the training set, a higher connectivity usually leads to faster learning, and fewer independent parameters, but it also results in unbounded fan-in and depth. Strictly layered architectures with restricted connectivity, on the other hand, need more epochs to learn and use more parameters, but generate more regular structures, with smaller, limited fan-in and significantly smaller depth (propagation delay), and may be better suited for VLSI implementations. When the number of inputs or outputs is not very small compared to the size of the training set, however, a strictly layered topology is seen to yield an overall better performance.

---

* This work was supported in part by NSF Grant MIP 90-13013.
Abbreviated version of this report is in print in the IEEE Transactions on Neural Nets, vol. 5, no. 11, Nov. 1994

I Introduction

Algorithms like the Cascade Correlation [1] or the Continuous ID3 (CID3, which is derived from a machine learning paradigm [2]), incrementally build a neural net. These algorithms take the guess work out of the training process: there is no need to guess the number of hidden units or their connections ahead of time. The algorithms themselves add units and connections as required, during the training process. However, these algorithms can lead to unrestricted fan–in and large depth (propagation delay). Also, the connections generated are irregular and unsuitable for VLSI implementation (an efficient VLSI implementation requires regularity and local interconnects). In this paper, we propose a modified version of the Cascade Correlation algorithm which leads to restricted fan–in, more regular connections and significantly smaller depth.

We begin with a brief introduction to the Cascade Correlation algorithm [1]. In this method, all the inputs and outputs are directly connected first and these connections are trained to minimize the error (i.e., squared error summed over all outputs over all I/O patterns). Hidden units are then installed one by one, incrementally reducing the error, till the desired error bound is met. Each unit is installed in two steps. In step 1, all the inputs and previously installed hidden units are connected to the input of the new hidden unit. Its output is not yet connected anywhere in the net. The input side connections of the new unit are then trained to *maximize* the correlation between its output and the residual error at the network outputs. This can be thought of as a maximal "alignment" with the residual error. The input side connection weights are frozen hereafter. In the 2nd phase of installing the hidden unit, its output is connected to all the network output units. All the connections feeding the output units are then trained to *minimize* the error. This can be thought of as "canceling" the error as much as possible by exploiting the "alignment" accomplished in step 1. This process is continued, and new units are added until the desired error bound is met.

This turns out to be an extremely efficient, very flexible, and fast training procedure. However, each new unit in effect adds a new layer, which leads to a very deep structure with a long propagation delay. It should be noted that the presence of layer(s) skipping or shortcut connections *does not reduce* the propagation delay because the delay is proportional to the *longest* path through the network which is unaffected by the presence of shortcut connections. The propagation delay is an important performance parameter especially for high-speed applications. Moreover, learning is likely to occur infrequently and perhaps off-line. Hence, it is worthwhile to spend more time in learning if the resultant net has a much smaller propagation delay. Another drawback of the Cascade Correlation algorithm is that the fan–in of the hidden and output units keeps on increasing as more units are added. The resulting structure with irregular connections and unbounded fan–in might be hard to implement in VLSI.

Attempts have been made to reduce the connectivity [3] of the Cascade Correlation algorithm. Smotroff et. al. [3] used "iterative atrophy" to prune less important connections. They came up with a saliency measure for each weight and used saliency windows to eliminate non-useful connections. Such an approach becomes computation intensive. Estimating the correct saliency for each weight can itself be a non-trivial task. This type of algorithm therefore spends effort first in finding all the connection weights and then in trying to undo a part of what it did and remove the non-salient connections. Another drawback of weight-pruning by saliency is that it does not lead to any improvement in the depth. In fact, the use of saliency windows would imply that the nets derived by weight pruning can be as deep as those obtained by the original version.

We have approached the problem differently and modified the Cascade Correlation algorithm to yield architectures with restricted fan-in and much smaller depth. An obvious way of controlling the fan–in and depth is to generate a "strictly layered" structure. By strictly layered, it is meant that there are no connections that skip a layer. Thus, the fan–in of a unit is limited to the number of units in the previous layer. If the number of units in each layer is sufficiently large, the number of layers needed (i.e., the depth) is usually very small as well. Next, we describe the algorithm(s) that generate the various layered topologies.

II The Training Procedure

Step 1. Train the biases of the output units as best as possible, to minimize the error. Note that the inputs and outputs are not connected at this time. This is illustrated in Figure 1-a.

Step 2. Install the first hidden layer, adding one unit at a time. Hidden units in this layer receive connections only from the input units and are not connected to other hidden units.

To install a hidden unit, it is connected to all of the network inputs and the connections are trained to *maximize the correlation* between its output and the residual error. Note that all the weights (input as well as output weights) associated with previously installed hidden units are held fixed when the input connections of the new unit are being trained. As a result, the new unit sees smaller residual error (at installation time) than previous units, because the previous units have already reduced the total error. The input–side weights of the new hidden unit remain frozen hereafter, as in the original Cascade Correlation algorithm.

In the second phase of installing a unit, its output is connected to all the network output units. All the fan–in connections of the output-layer units (those emanating from previously installed hidden units as well as those connected to the hidden unit being currently installed), and their biases are then trained to *minimize the error.*

In order to restrict the fan–in and achieve regularity, the number of units in each layer is held fixed at a predetermined constant which is evaluated via heuristics and trial-and-error, depending on the problem at hand. Thus, all hidden layers (except possibly the last) have the same number of units. The creation of the first layer is illustrated in Figure 1-b.

Note that just one layer of hidden units (two layers of adjustable weights) should be sufficient to approximate any static input/output mapping, to the degree desired, since every additional unit *reduces* the residual error and the process of adding new units can be continued till the desired tolerance is achieved. This is analogous to the fact that any Boolean or Switching function can be implemented with two levels of logic in Sum-of-Products or Product-of-Sums formats. Such a single-hidden-layered implementation might, however, lead to arbitrarily large output fan–in and perhaps an excessively large number of hidden units.

Step 3. Collapse the output layer into the next hidden layer. This means that the N units which were output units so far, are now deemed to be the first N units in the next hidden layer. The old output layer thus becomes a part of the next hidden layer. A *new* output layer is created and all the previous output units (which are now a part of the new hidden layer) are connected to all the new output units. These connections are trained to minimize the error. This step is depicted in Figure 1-c.

The motivation for utilizing the old output units and their connections in this manner is threefold. First, this reduces the training time. Second, the set of weights feeding these (previous output, now hidden) units are repeatedly trained many times, during the creation of the previous (just completed) hidden layer. Hence, the possibility of incorporating suboptimal and wasteful connections is reduced. Third, this facilitates good initial guesses of weights connecting this layer to the new output layer as explained next.

Denote the previous output units by $1, 2, \cdots, i, \cdots, N$; the new output units by $1', 2', \cdots, i'$,
$\cdots N'$; and the weight of the link from unit j to unit i' by $W_{ji'}$. The biases of all the new output units are initialized to 0. The magnitude of weight $W_{ji'}$ is set proportional to the correlation between the output of unit j and the target values of the ith network output. If the magnitude is above a preset threshold, the sign is set positive, otherwise the sign is set negative. This way, if the ith unit had produced correct outputs for most patterns, then $W_{ii'}$ (weight of the link from unit i to the *corresponding* output unit i') would be initialized to a large positive value, which would favor unit i' to turn *on* whenever unit i is *on* . On the other hand, if the ith unit produced wrong outputs for most patterns, then $W_{ii'}$ would be initialized to a large negative value, which would favor unit i' to turn *off* whenever unit i is *on* . Thus, this choice of initial values leads to a reduction in the output error. It was found to result in faster convergence in most cases.

Step 4. Expand the newly formed hidden layer by adding more hidden units one at a time, exactly as in step 2.

Step 5. Repeat steps 3 and 4 creating new layers until the training is successful or until a predetermined number of iterations where the trial is declared unsuccessful and is abandoned.

It should be emphasized that creating a layered topology is just one of a large number of possible ways of limiting the depth and fan–in. Even the layered topology itself can have several different variations, each of which in turn can be generated in many different ways. Indeed the Cascade Correlation is a very flexible algorithm and it is not feasible to try out all possible modifications or even a significant fraction thereof. We have tested several possibilities and selected the above method for the purpose of illustration because it performs well and is relatively simple.

III Results

We have run this algorithm on a large number of problems. Comparison of the original Cascade Correlation and the layered version described above appears in Tables 1 to 5. We used 4 benchmarks viz. the Two Spirals, Sonar, Vowel and the NETTalk benchmarks from the CMU collection [4]. We ran 100 trials (10 sets of 10 trials each) with every algorithm, for the first 3 benchmarks and 10 trials with each algorithm for the NETTalk benchmark. The NETTalk nets are big and take a formidable amount of time to learn. The time required for running more than 10 trials is prohibitive. For each set, we used the same seed to initialize the random number generator in all algorithms. This means that for 10% of the runs (i.e., the first run in each set), all the algorithms started off from the same state of the random number generator.

Relevant parameters of interest are : maximum fan–in, number of units, total number of independent adjustable parameters (all weights and biases in the net), depth, number of epochs, number of connection crossings (please refer to [1] for definition and description of this parameter) and the number of bit errors on the test set. These parameters were stored for each of the runs, but only the values from successful runs were used to accrue the statistics. Maximum, minimum, average value and standard deviation were calculated for each of these parameters (independently of the other parameters). The results are shown in the tables. Also included is the number of bit errors on the training set after an unsuccessful run, where the statistics is meaningful only for unsuccessful runs.

(a) The Two spirals Benchmark [4, 5] : Table 1 shows the results for the Two Spirals classification problem where asymmetric sigmoidal units (output $\in$ $(0, 1)$) were used. The problem specification has 2 inputs and 1 output and 194 training patterns (97 points from each spiral).

Note that the table also shows a "loosely layered" version besides the strictly layered version. In the loosely layered version, all the units in a hidden layer receive connections from the original (external) inputs as well, besides the connections from the previous hidden layer. The motivation for adding the connections to the network inputs is twofold. First, note that in a strictly layered structure, the units in a layer "see" the inputs only through the previous layer. Here, it is conceivable that there could be a "loss of information" in the earlier (closer to input) layer(s) which may prevent the later (closer to output) layers from successfully learning the task. This may happen if, for example, a layer has too few units (or independent parameters) to capture all the features associated with the inputs or if certain combinations of weight values evolve during training. We encountered this situation in practice. It is intuitively clear that if the hidden units have access to the original external inputs, this kind of "irrecoverable loss of information" is not possible. Second, these connections considerably speed up the learning process, *without increasing the depth, while maintaining the restricted fan–in.* Note that the number of inputs is a *constant*. Adding connections to the inputs therefore increases the maximum fan–in by a constant amount. The fan–in is still independent of the number of hidden units installed, and is not unbounded as in the original version.

We used 15 units per hidden layer in this version. As mentioned above, the number of units (parameters) in a layer should be large enough to capture all the features of the inputs, otherwise there could be a loss of information leading to an unsuccessful run. After some trial and error, it was found that the strictly layered algorithm can converge if the number of units per layer is larger than 5. At this extreme, (5 units per layer) the algorithm uses too many layers and defeats the purpose of trying to limit the depth. On the other hand, we wanted to limit the maximum fan–in to about half that of the nets generated by the original version. It was therefore decided to use 15 units per layer.

As seen in the table, the maximum fan-in (average value accrued over 100 trials) for the original Cascade Correlation algorithm is 32.2 ; while that for the strictly layered version is 15 ; and for the loosely layered version, it is 17. Thus, the fan-in is reduced by about $\frac{1}{2}$. Similarly, the depths (average) for the layered versions are 6.3 and 5.2 while the depth generated by the original version is 31.2. This demonstrates a significant reduction in the depth or the propagation delay (about 5 times smaller depth). The number of units used by the layered versions, on the other hand, increases by a factor between 2 and 2.5. However, the total number of independent parameters (weights and biases) is a better measure of the overall complexity than the number of units alone, because it incorporates both the number of units and their connectivity. It is seen that the strictly layered version uses about 1.6 times; and the loosely layered version uses about 1.37 times the number of parameters used by the original algorithm.

Table 2 shows the results for the same benchmark, but with symmetric sigmoidal units (output $\in (-0.5, +0.5)$). The original version shows significant improvement over the case with asymmetric sigmoidal units. The strictly layered version does not show such an improvement. The layered version which allows connections to input units does, however, show a substantial improvement. As mentioned above, in the strictly layered version it is possible to encounter an irrecoverable loss of information in a layer, if the number of hidden units is too small or if certain combinations of weight values evolve during training. This indeed was the case in a few of the unsuccessful trials.

There can be other modes of failure too. In some cases, running the program longer might have produced a successful run but only with an excessively large number of units and a depth and fan–in larger than that of the original version. The table shows that when the number of units per layer is small, the success rate for the strictly layered version is low and the average depth of the nets is even larger than that of the original version (success rate of 8% and average depth ≈ 19 for 7 units per layer). As the number of units per layer is increased, the success rate of the strictly layered version improves. The depth of the nets also reduces considerably (by about half). Because the strictly layered version failed several times in this particular case, this table has an extra row : viz., the number of bit errors on the training set after an unsuccessful trial. This residual number of erroneous bits falls as the number of units per layer is increased.

It was also observed that the bit errors are not always in synch with the sum squared error in the strictly layered version. Especially as the number of layers grows, a newly installed unit reduces the sum squared error by a small amount but the number of bit errors *increases* instead of decreasing. The later units have to undo this and *reduce* the number of bit errors. (Training is successful only if there are *no* bit errors). These oscillations can waste too many units and even layers.

As mentioned above, this kind of "irrecoverable loss of information" can be avoided in a loosely layered version, wherein the hidden units are connected to the inputs (original external inputs). Nor will the oscillatory behavior be so prolonged and detremental, if the network inputs are available to the unit(s) being installed. Table 2 indicates that the loosely layered topology performs exactly as expected: the algorithm does not get stuck, oscillate or fail in any other way with this topology as evidenced by the 100% success rate. Moreover, the algorithm converges faster and uses fewer parameters than the strictly-layered version; but is not as fast as the original version where all the connections are present. The regularity in the strictly layered version is now broken because of the connections to the input units. However, note that the maximum fan–in is still fixed; it does not grow arbitrarily as in the original version. Also, the depth is much smaller than that generated by the original version.

Cios and Liu [2] have also used the two spirals benchmark to illustrate their CID3 algorithm. The net reported in [2] has 30 hidden units distributed in 5 hidden layers(depth = 6), a maximum fan–in of 32, and requires 431 independent parameters. Thus, the number of parameters utilized by CID3 is smaller than that utilized by the loosely layered version(527), but the maximum fan–in is about double that of the loosely layered version(17). The depths of the generated nets are comparable. In fact, the CID3 has some of the same drawbacks as the Cascade Correlation algorithm: it connects all previously installed hidden layers to the new layer being created, which leads to an unbounded fan–in and irregular connections.

The spirals used to generate Table 1 had a radius $R = 2.0$ and were centered at (2,2) in the xy plane. Consequently, they were contained in the square with vertices (0,0), (0,4), (4,4), (4,0) which lies entirely in the 1st quadrant. The net reported in [2], on the other hand, was trained on spirals centered at the origin. For a proper comparison with CID3, we therefore used spirals centered at the origin (0,0) with a radius $R = 6.5$. This way the x and y coordinates of the points on the spirals take both positive and negative values and are symmetric with respect to the origin. For this training data, the loosely layered version of our algorithm generated nets with 7 hidden units per layer, with an average of 6 hidden layers (depth = 7), and utilized 436 independent parameters on the average. These nets are better than the net generated by the CID3 algorithm since the depth and number of independent parameters used is about the same, but the the maximum fan–in is restricted to the fixed value 9 (about $\frac{1}{3}$rd), and the connections are more regular.

(b) The Sonar Benchmark [4, 6] : Table 3 shows the results for the sonar benchmark. Here, the inputs are all positive numbers. Consequently, only asymmetric sigmoidal units were used. This problem has a large number of input units (60). Also, the number of hidden units used is usually very small. The maximum fan–in is therefore dominated by the number of input connections. For the layered version, just one hidden layer with a small number of hidden units was sufficient to achieve the desired classification in each of the 100 trials. Because of such a small depth, and the 100% success rate, it is not necessary to try the loosely layered version where connections to the input units are allowed. The table therefore has only two columns: one for the original cascade correlation and the other for the layered version. Even the subcolumns under the column for strictly layered version which correspond to different number of units per layer are unnecessary, since the number of units used is always small enough to fit in a single layer.

(c) The Vowel Benchmark [4, 7] : Table 4 shows the results for the Vowel Benchmark with symmetric sigmoidal units (output $\in [-0.5, +0.5]$). The net has 10 inputs and 11 outputs, where exactly one out of the 11 output units is "on" at a time, to indicate the spoken vowel. This corresponds to a localized output encoding. As seen in the table, the number of hidden units employed is usually very small (in fact, smaller than the number of input or output units). The maximum fan–in is therefore dominated by the number of input connections. Just one hidden layer with a small number of hidden units was found to be sufficient for this task. Hence, is not necessary to try the loosely layered version. Note that the total number of parameters(average value) used by the layered version is significantly smaller than that of the original version.

(d) The NETTalk benchmark [4, 8] : Table 5 shows the results for this benchmark for which there are 196 input and 26 output units. We randomly chose a set of 200 words (from the 1000 available in the CMU version), which generated 1114 training patterns. Here, the fan-in is dominated by the number of input units. It it interesting to note that the strictly layered version uses a much larger number of hidden units but a smaller total number of independent parameters. This is due to the restricted connectivity. Also, the depth is much smaller than that generated by the original version (about $\frac{1}{6}$th). An examination of the nets revealed that the direct input-output connections in the original version is the single most dominant contributor to the total number of independent parameters ($196 \times 26 = 5096$ or about 50% of the total number of parameters. Please refer to Table 3). Since the strictly layered version performs better without these connections, there is no need to try the loosely layered version where these connections are present.

IV Discussion and Conclusions

Our results indicate that the layered version always requires more hidden units. This is expected, since the unit being installed has fewer connections. It can only see units in the previous hidden layer and thus has less information to work with than a unit in the original version, which is connected to all the previous units and network inputs.

The number of independent parameters, which is a better measure of the overall network complexity, however, shows an interesting behavior. When the number of training patterns is large compared to the number of inputs or outputs, the original version utilizes fewer parameters than the layered version. This is illustrated by the two spirals benchmark where the number of input units is two, and the number of I/O pairs to be learned is 194 or about 100 times larger. For such problems, there is a tradeoff between connectivity and other performance attributes. The original version with full connectivity is at one extreme. It yields faster convergence and a smaller number of parameters but also leads to very deep nets with arbitrarily large fan–in. The strictly layered version is at the other extreme and requires more parameters and longer learning time, but yields restricted fan–in and much smaller depth. The loosely layered version falls in between these extremes, trading off more connectivity for fewer parameters and equal or smaller depth, and may be the best compromise.

When the input or output dimensionality is not so small compared with the size of the training set, the layered version utilizes fewer parameters and yields overall better performance. This is illustrated by the Vowel and the NETTalk benchmarks. The ratio of the number of training patterns to the number of input units is about 9 and 5, for the Vowel and NETTalk benchmarks, respectively. In such cases, connecting the network inputs to the outputs results in a large (in fact larger than required) number of connections. Furthermore, training all of them simultaneously turns out to be less efficient than incrementally introducing hidden units and training the small number of associated connections at a time.

In conclusion, we have proposed a modified version of the Cascade Correlation algorithm which controls the connectivity of the units being added to restrict the fan–in and generates layered nets with a very small depth and regular connections. Our data illustrates the tradeoffs between connectivity and other performance attributes.

Several future extensions are possible. For instance, the number of units per layer need not be fixed. It can be decided dynamically and the creation of a new layer can be started if the error reduction achieved by expanding the layer begins to taper off. Another approach is to stick to the strictly layered structure as far as possible and only occasionally install units with direct (layers skipping) connections to the network inputs, when there is little or no reduction in the error.

Preliminary analysis of the generalization characteristics of these nets has been done. Even though the layered version utilizes a larger number of parameters, its generalization performance appears to be comparable to or better than the original version, as illustrated by the number of bit errors on the test sets for the Spirals and Vowel benchmarks in Tables 1 and 2. However, further investigation is necessary to draw any conclusions.

References

[1] S. E. Fahlman and C. Lebiere, "The Cascade Correlation Learning Architecture," in *Neural Information Processing Systems 2* (D. S. Touretzsky, ed.), pp. 524–532, Morgan Kaufman,

1990.

[2] K. J. Cios and N. Liu, "A Machine Learning Method for Generation of a Neural Network Architecture: A continuous ID3 Algorithm," *IEEE Transactions on Neural Networks*, vol. 3, pp. 280–291, Mar. 1992.

[3] I. G. Smotroff, D. H. Friedman, and D. Connolly, "Large Scale Networks Via Self Organizing Hierarchical Networks," in *Proceedings of the SPIE conference on applications of AI and Neural Networks*, April 1991.

[4] Fahlman, S. E. et. al., *Neural Nets Learning Algorithms and Benchmarks Database.* maintained by S. E. Fahlman et. al. at the Computer Science Dept., Carnegie Mellon University.

[5] K. J. Lang and M. J. Witbrock, "Learning to Tell Two Spirals Apart," in *Proceedings of the 1988 Connectionist Models Summer School, San Mateo, CA* (D. Touretzsky, G. Hinton, and T. Sejnowski, eds.), Morgan Kaufman Publishers, 1988.

[6] R. P. Gorman and T. J. Sejnowski, "Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets," *Neural Networks*, vol. 1, pp. 75–89, 1988.

[7] A. J. Robinson and F. Fallside, "A Dynamic Connectionist Model for Phoneme Recognition," in *Proc. of nEuro, June 1988, Paris*, Jun. 1988.

[8] T. J. Sejnowski and C. R. Rosenberg, "Parallel Networks That Learn to Pronounce English Text," *Complex Systems*, vol. 1, pp. 145–168, 1987.

Parameter ↓		⟵ Algorithm Type ⟶		
		Original Cascade Correlation	Strictly Layered units per layer 15	Loosely Layered units per layer 15
Maximum Fan–in	max	40	15	17
	min	25	15	17
	average	32.2	15	17
	std. dev.	3.36	0	17
Number of Hidden Units	max	38	108	73
	min	23	46	41
	average	30	72	55
	std. dev.	3.36	11.2	6.12
Total number of Independent Parameters	max	858	1537	1105
	min	348	543	527
	average	570	962	784
	std. dev.	113.8	179.2	111.1
Depth	max	39	9	6
	min	24	5	4
	average	31.2	6.3	5.2
	std. dev.	3.36	0.8	0.44
Number of Epochs	max	8717	8011	17643
	min	2460	4008	3791
	average	3337	5828	5134
	std. dev.	768.8	779.3	1720.5
Connection Crossings ($\times 10^7$)	max	10.4	15.1	23.7
	min	4.04	5.29	5.48
	average	6.78	9.51	8.1
	std. dev. (percentage of the mean)	19.87%	18.6%	23.9%
Bit errors on Test Set	max	30(15.5%)	27(13.9%)	17(8.8%)
	min	5(2.6%)	4(2.1%)	3(1.5%)
	average	15.4(7.9%)	13.4(6.9%)	8.9(4.6%)
	std. dev.	5.6	4.8	3.1

Table 1: The Two Spirals benchmark [4, 5] with Asymmetric Sigmoidal units (output $\in (0,1)$). The problem specification has 2 inputs and 1 output.

Parameter ↓		⟵ Algorithm Type ⟶					
		Original Cascade Correlation	Strictly Layered			Layered + Input unit connections		
			units per layer			units per layer		
			7	10	12	7	10	12
Success rate		100 %	8 %	48 %	69 %	100 %	100 %	100 %
Maximum Fan–in	max	24	7	10	12	9	12	14
	min	14	7	10	12	9	12	14
	average	17.35	7	10	12	9	12	14
	std. dev.	1.98	0	0	0	0	0	0
Number of Hidden Units	max	22	168	164	151	46	41	44
	min	12	92	63	66	27	27	30
	average	15.35	123.12	108.04	95.2	34.87	35.21	36.52
	std. dev.	1.98	23.19	26.42	20.14	3.08	2.4	2.75
Total No. of Independent Parameters	max	322	1317	1729	1851	418	437	527
	min	117	703	617	745	230	261	315
	average	176.47	954.25	1113.75	1123.75	306.98	366.44	414.64
	std. dev.	38.75	187.18	290.33	261.83	30.39	31.94	39.23
Depth	max	23	25	18	14	8	6	5
	min	13	15	8	7	5	4	4
	average	16.35	19.13	12.38	9.51	6.37	4.95	4.39
	std. dev.	1.98	3.09	2.69	1.69	0.56	0.26	0.49
Number of Epochs	max	2402	10172	10406	9208	10080	3180	3477
	min	1332	5930	4193	4761	2042	2129	2383
	average	1743.99	7578.75	7113.46	6536.96	2640.1	2741.15	2964.97
	std. dev.	205.63	1325.84	1497.23	1148.76	175.8	190.24	175.80
Connection Crossings ($\times 10^7$)	max	3.5	11.34	16.26	15.87	3.49	4.15	5.04
	min	1.25	5.98	5.28	6.78	1.93	2.47	3.09
	average	1.95	8.13	10.04	10.33	2.63	3.38	4.06
	std. dev.	0.42	1.67	2.67	2.37	0.27	0.31	0.34
Bit errors on Test set	max	20(10.3%)	66(34%)	61(31.4%)	51(26.3%)	24(12.4%)	21(10.8%)	22(11.3%)
	min	0(0%)	42(21.6%)	22(11.3%)	16(8.2%)	7(3.6%)	5(2.5%)	3(1.5%)
	average	9.22(4.7%)	48.5(25%)	38.5(19.8%)	31.5(16.3%)	14.1(7.3%)	12(6.2%)	11.6(6%)
	std. dev.	4.16	7.58	8.71	7.13	4.10	3.44	3.41
Bit errors on Training Set (Unsuccessful trials only)	max	not applicable	53(27.3%)	20(10.3%)	14(7.2%)	n. a.	n. a.	n. a.
	min		1(0.5%)	1(0.5%)	1(0.5%)			
	average		20.7(10.7%)	6.0(3.1%)	4.1(2.1%)			
	std. dev.		11.1	4.3	3.7			

Table 2: The Two Spirals benchmark [4, 5] with Symmetric Sigmoidal units (output $\in (-0.5, +0.5)$).

Parameter ↓		⟵ Algorithm Type ⟶	
		Original Cascade Correlation	Strictly Layered Version
Success rate		100 %	100 %
Maximum Fan–in (Dominated by number of input units = 60)	max	64	60
	min	62	60
	average	62.8	60
	std. dev.	0.51	0
Number of Hidden Units (all in a single layer for layered version)	max	3	7
	min	1	4
	average	1.8	5.14
	std. dev.	0.51	0.86
Total number of Independent Parameters	max	250	435
	min	123	249
	average	173.45	319.68
	std. dev.	32.4	53.62
Depth	max	4	2
	min	2	2
	average	2.8	2
	std. dev.	0.51	0
Number of Epochs	max	713	786
	min	359	378
	average	538.12	535.96
	std. dev.	83.3	80.2
Connection Crossings	max	15963688	21232016
	min	5891808	11377496
	average	9864221	16256886
	std. dev.	2045699	2497822
Bit errors on Test Set	max	34(32.7%)	37(35.6%)
	min	17(16.3%)	14(13.5%)
	average	23.8(22.9%)	25.2(24.3%)
	std. dev.	3.5	5.1

Table 3: The Sonar benchmark [4, 6] with Asymmetric Sigmoidal units (output $\in (0,1)$).

Parameter ↓		⟵ Algorithm Type ⟶ Original Cascade Correlation	Strictly Layered Version
	max	12	10
Maximum Fan–in (Dominated	min	12	10
by number of input units = 10)	average	12	10
	std. dev.	0	0
	max	1	6
Number of Hidden Units	min	1	4
(all in a single layer	average	1	4.84
for the layered version)	std. dev.	0	0.42
	max	143	143
Total number of	min	143	99
Independent Parameters	average	143	117
	std. dev.	0	9.23
	max	2	2
Depth	min	2	2
	average	2	2
	std. dev.	0	0
	max	406	579
Number of Epochs	min	202	274
	average	287	423
	std. dev.	36.96	55.98
	max	2.34	1.82
Connection Crossings ($\times 10^6$)	min	1.11	0.88
	average	1.61	1.36
	std. dev. (percentage of the mean)	13.5%	13.2%
	max	166(31.4%)	100(18.9%)
Bit errors on	min	143(27.1%)	69(13.1%)
Test Set	average	158.8(30.1%)	80.8(15.3%)
	std. dev.	4.4	6.0

Table 4: The Vowel benchmark [4, 7] with Symmetric Sigmoidal units, (output $\in (-0.5, +0.5)$). The problem specification has 10 inputs and 11 outputs. A localized output encoding is used, i.e., only one of the 11 output units is "on" at a time to indicate the class of the vowel spoken.

Parameter ↓		⟵ Algorithm Type ⟶ Original Cascade Correlation	Strictly Layered Version (35 units per layer)
Maximum Fan–in (Dominated by number of input units = 196)	max	226	196
	min	221	196
	average	223	196
	std. dev.	1.49	0.0
Number of Hidden Units	max	29	132
	min	24	100
	average	25.7	104.8
	std. dev.	1.49	8.71
Total number of Independent Parameters	max	11995	11115
	min	10750	10041
	average	11172	10265
	std. dev.	371.9	285.8
Depth	max	30	5
	min	25	4
	average	26.70	4.08
	std. dev.	1.49	0.29
Number of Epochs	max	5080	5812
	min	4340	5300
	average	4641	5565
	std. dev.	194.9	179.4
Connection Crossings ($\times 10^9$)	max	21.3	6.84
	min	17.5	6.43
	average	19.1	6.64
	std. dev. (percentage of the mean)	5.6%	2%

Table 5: The NETTalk benchmark [4, 8] with Symmetric Sigmoidal units. There are 196 inputs and 26 outputs.

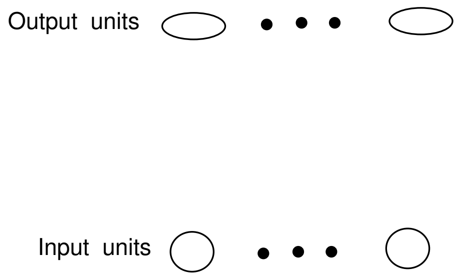

Figure 1−a : Step 1. Train the biases of the output units. Note that the inputs are not yet connected.

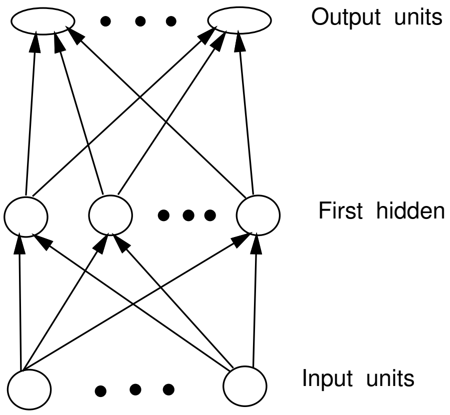

Figure 1−b : Step 2. Install the first hidden layer. Units are added one at a time. Input side connections are trained first. Once trained, the input side weights remain fixed. Output side connections for the new unit are then installed and all the output connections are trained .

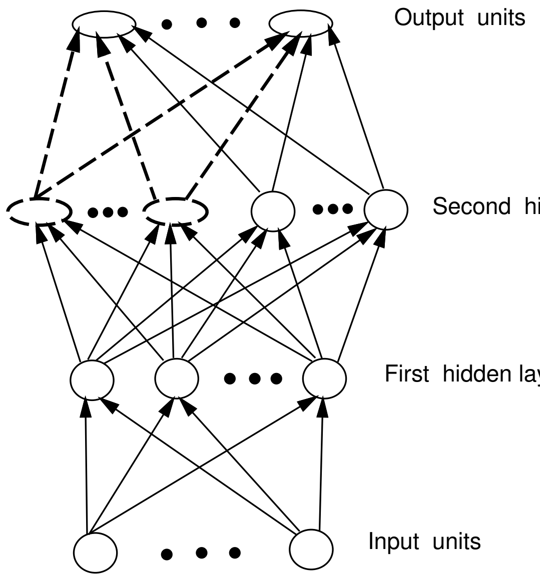

Figure 1−c : Steps 3 and 4. Install the second (and successive) hidden layers.

Units in the old output layer (shown by dotted ellipses) are collapsed into the next (second) hidden layer. A new output layer is created. Input side weights of the newly collapsed hidden layer remain fixed. The output side connections of the newly formed hidden layer (shown in dotted linestyle) are all trained simultaneously. Then the hidden layer is further expanded by installing more units, one at a time, as in step 2.