C Appendix

This appendix contains all the data from Experiment 2 described in Section 5.5. Table 12 and 13 show the most important figures and the percentages of correctly classified test patterns for the HCCA and the 2CCA network, respectively. The first three columns show the number of asymmetric sigmoid, symmetric sigmoid, and Gaussian units installed for each of the ten runs. "Euclid" ("Bin") refers to the Euclidean (Binary) method, respectively, of measuring the performance, and the "Avg." row shows the average for all the columns.

Experiment 2: Improving the Learnability (Part 1)							
	HCCA						
Run	*asigm*	*ssigm*	*gauss*	N_{hid}	N_{epoch}	Perform(Euclid)	Perform(Bin)
1	1	1	19	21	1987	84.4	81.6
2	2	0	19	21	2009	83.5	82.3
3	4	0	19	23	2182	77.7	75.8
4	2	2	17	21	2149	84.6	80.8
5	2	1	17	20	1953	85.2	83.5
6	4	2	19	25	2564	78.7	76.5
7	3	0	21	24	2310	77.5	74.2
8	1	2	21	24	2403	77.6	76.3
9	4	1	20	25	2497	75.4	73.6
10	2	2	21	25	2473	80.3	77.1
Avg.	25	11	193	22.9	2252	80.5	78.2

Table 12: Experiment 2: Improving the Learnability for HCCA networks trained on S^{500}_{train}

Experiment 2: Improving the Learnability (Part 2)							
	2CCA						
Run	*asigm*	*ssigm*	*gauss*	N_{hid}	N_{epoch}	Perform(Euclid)	Perform(Bin)
1	17	2	20	39	3818	93.3	91.3
2	12	3	27	42	3808	92.1	89.2
3	8	4	15	27	2815	92.8	90.4
4	19	3	23	45	4243	93.6	91.5
5	13	3	21	37	3612	93.3	90.4
6	14	6	19	39	3883	92.5	90.2
7	9	2	25	36	3657	92.2	89.6
8	10	3	19	32	3442	92.5	89.9
9	9	2	17	28	2880	93.4	90.4
10	20	2	26	48	4108	92.8	90.6
Avg.	131	30	212	37.3	3627	92.9	90.4

Table 13: Experiment 2: Improving the Learnability for 2CCA networks trained on S^{500}_{train}

Experiment 1: Scaling Behavior (Part 5)								
	HCCA				2CCA			
Run	N_{epoch}	N_{hid}	Perf(Euclid)	Perf(Bin)	N_{epoch}	N_{hid}	Perf(Euclid)	Perf(Bin)
1	3432	34	85.4	82.9	7913	88	95.3	92.0
2	3272	31	85.1	83.7	8673	104	94.5	91.7
3	3200	29	88.4	87.0	6811	72	95.8	93.0
4	2868	27	89.1	86.2	5471	55	94.7	91.7
5	4171	42	84.8	83.9	6960	76	94.7	91.9
6	3708	35	86.4	84.0	9178	110	95.3	93.5
7	3951	36	87.1	84.9	8655	103	95.4	93.6
8	3506	33	86.2	85.3	8598	96	95.4	93.4
9	4018	39	85.2	83.6	8418	101	95.9	93.7
10	3940	39	85.3	86.6	6618	71	94.0	91.7
Avg.	3606	34.5	86.3	84.8	7730	87.6	95.1	92.6

Table 10: Experiment 1: Scaling Behavior for training set S^{750}_{train}

Experiment 1: Scaling Behavior (Part 6)								
	HCCA				2CCA			
Run	N_{epoch}	N_{hid}	Perf(Euclid)	Perf(Bin)	N_{epoch}	N_{hid}	Perf(Euclid)	Perf(Bin)
1	3918	39	88.4	87.8	9370	111	96.8	95.1
2	4170	42	88.9	86.8	9336	115	97.6	95.2
3	4192	42	88.4	86.1	8193	93	97.2	95.3
4	3578	35	89.2	87.1	9662	124	97.0	95.4
5	4573	46	86.1	84.4	10071	114	97.8	95.4
6	3910	37	87.4	86.3	9841	125	97.5	95.8
7	4044	41	88.8	87.2	10053	126	96.5	95.0
8	5073	53	88.4	86.7	8222	92	97.4	94.9
9	3779	37	88.2	86.7	9749	117	97.1	94.3
10	3705	35	91.0	89.1	11356	140	98.1	95.7
Avg.	3994	40.7	88.5	86.8	9585	115.7	97.3	95.2

Table 11: Experiment 1: Scaling Behavior for training set S^{1000}_{train}

Experiment 1: Scaling Behavior (Part 3)								
	HCCA				2CCA			
Run	N_{epoch}	N_{hid}	Perf(Euclid)	Perf(Bin)	N_{epoch}	N_{hid}	Perf(Euclid)	Perf(Bin)
1	2724	24	76.5	74.5	4424	42	90.4	87.5
2	2501	23	80.6	78.7	4462	45	88.3	84.2
3	2686	22	79.6	78.0	4035	38	89.4	86.7
4	2875	25	78.9	76.8	4025	34	89.4	86.2
5	2398	21	82.7	79.6	4070	39	89.6	86.6
6	2686	25	81.7	79.5	4490	45	89.4	85.0
7	2820	25	77.1	75.2	4924	48	88.5	85.1
8	2657	26	79.8	76.6	5243	54	89.3	86.4
9	2531	22	75.6	74.5	5189	56	87.9	84.3
10	2786	26	79.9	77.6	5292	57	90.8	88.2
Avg.	2666	23.9	79.2	77.1	4615	45.8	89.3	86.0

Table 8: Experiment 1: Scaling Behavior for training set S^{300}_{train}

Experiment 1: Scaling Behavior (Part 4)								
	HCCA				2CCA			
Run	N_{epoch}	N_{hid}	Perf(Euclid)	Perf(Bin)	N_{epoch}	N_{hid}	Perf(Euclid)	Perf(Bin)
1	2800	25	86.1	84.4	6756	76	93.1	90.1
2	3088	27	86.9	84.6	5072	51	91.0	88.6
3	3158	27	86.1	84.1	5552	59	92.5	90.1
4	3030	28	84.1	82.0	6238	75	91.9	87.9
5	3015	27	82.8	80.8	5285	59	92.9	89.2
6	2808	25	85.8	84.4	5492	59	92.4	89.4
7	3298	30	85.2	82.9	5620	62	91.6	89.4
8	3032	26	84.2	82.4	4937	49	92.2	89.6
9	3152	30	85.8	82.8	4777	48	91.5	89.0
10	2616	24	86.4	82.9	6286	66	92.4	90.1
Avg.	2999	26.9	85.3	83.1	5601	60.4	92.2	89.4

Table 9: Experiment 1: Scaling Behavior for training set S^{500}_{train}

B Appendix

This appendix contains all the data from the scaling experiments described in Section 5.4. Table 6 to 11 show the most important figures and the percentages of correctly classified test patterns for the HCCA and the 2CCA networks.

"Euclid" ("Bin") refers to the Euclidean (Binary) method, respectively, of measuring the performance, and the "Avg." row shows the average for all the columns.

Experiment 1: Scaling Behavior (Part 1)								
	HCCA				2CCA			
Run	N_{epoch}	N_{hid}	Perf(Euclid)	Perf(Bin)	N_{epoch}	N_{hid}	Perf(Euclid)	Perf(Bin)
1	1649	13	72.1	68.8	3317	30	79.3	77.0
2	1627	13	71.4	69.5	3354	32	79.4	76.0
3	1703	15	68.1	65.3	3360	34	75.5	71.8
4	1955	17	70.3	68.7	3374	31	77.3	74.4
5	1608	13	73.5	71.2	2955	26	79.8	75.8
6	1751	14	69.8	67.1	3805	41	78.8	75.5
7	1567	13	67.6	64.5	3126	28	74.9	71.2
8	1873	16	62.5	60.7	2845	25	79.6	77.3
9	1776	15	74.0	72.1	2765	22	79.3	76.4
10	2012	16	69.2	66.6	3329	31	81.5	79.3
Avg.	1752	14.5	70.0	67.5	3223	30.0	78.5	75.5

Table 6: Experiment 1: Scaling Behavior for training set S^{100}_{train}

Experiment 1: Scaling Behavior (Part 2)								
	HCCA				2CCA			
Run	N_{epoch}	N_{hid}	Perf(Euclid)	Perf(Bin)	N_{epoch}	N_{hid}	Perf(Euclid)	Perf(Bin)
1	2025	16	79.1	76.9	3557	30	90.1	86.6
2	2229	19	76.8	74.1	3836	37	88.2	84.5
3	2339	21	74.6	72.5	3698	37	89.2	85.8
4	2098	18	78.3	76.5	4174	41	87.4	84.1
5	2224	19	80.8	78.6	4844	51	88.9	86.3
6	1964	17	78.4	76.1	3896	36	86.3	83.2
7	2047	17	75.7	73.8	3882	37	87.9	84.3
8	2432	22	80.9	78.8	3932	36	86.5	82.5
9	2101	19	79.8	77.2	3172	27	89.3	86.8
10	1875	16	76.0	73.9	3698	34	88.0	85.0
Avg.	2133	18.4	78.0	75.8	3868	36.6	88.2	84.9

Table 7: Experiment 1: Scaling Behavior for training set S^{200}_{train}

Evolution of a typical 2CCA network (Part 2)							
# hid.	N_{epoch}	EI	Corr.	E^{100}_{train}(Euclid)	E^{100}_{train}(Bin)	E^{100}_{test}(Euclid)	E^{100}_{test}(Bin)
21	2388	–	0.1540				
	2436	0.3039	–	4	6	18.9	22.8
22	2537	–	0.3305				
	2545	0.3019	–	4	7	19.9	22.7
23	2642	–	0.3278				
	2707	0.2834	–	2	6	18.8	22.7
24	2818	–	0.4642				
	2826	0.2810	–	2	5	18.9	22.8
25	3004	–	0.6083				
	3065	0.2440	–	2	5	17.9	21.1
26	3175	–	0.2852				
	3183	0.2424	–	3	5	17.6	21.5
27	3272	–	0.2611				
	3326	0.2285	–	1	4	17.3	21.2
28	3394	–	0.2575				
	3402	0.2266	–	1	4	17.1	21.4
29	3544	–	0.4695				
	3607	0.1997	–	1	3	17.6	21.4

Table 5: Evolution of a typical 2CCA network trained on S^{100}_{train} (Continued).

Evolution of a typical 2CCA network (Part 1)							
# hid.	N_{epoch}	EI	Corr.	E^{100}_{train}(Euclid)	E^{100}_{train}(Bin)	E^{100}_{test}(Euclid)	E^{100}_{test}(Bin)
	37	0.9262	–	54	76	59.0	79.4
1	99	–	0.7630				
	138	0.8324	–	47	59	53.0	63.5
2	186	–	0.6156				
	231	0.7592	–	44	56	50.9	61.9
3	308	–	0.6158				
	359	06900	–	34	43	43.1	54.4
4	424	–	0.5161				
	465	0.6614	–	30	40	38.4	50.0
5	523	–	0.2944				
	558	0.6407	–	25	43	35.2	45.4
6	624	–	0.3094				
	632	0.6375	–	27	40	34.7	44.2
7	703	–	0.3697				
	729	0.6183	–	24	38	33.4	42.1
8	806	–	0.3840				
	852	0.5853	–	17	34	27.3	40.2
9	927	–	0.3772				
	1000	0.5440	–	17	29	28.8	36.2
10	1071	–	0.2578				
	1079	0.5381	–	17	29	27.5	36.3
11	1138	–	0.2249				
	1173	0.5127	–	16	26	24.8	33.6
12	1245	–	0.3012				
	1253	0.5093	–	14	24	25.0	32.8
13	1309	–	0.2864				
	1332	0.4899	–	14	24	24.1	31.9
14	1387	–	0.2544				
	1395	0.4861	–	14	24	24.5	31.4
15	1480	–	0.2825				
	1545	0.4572	–	11	18	24.5	30.8
16	1651	–	0.3917				
	1659	0.4438	–	10	14	23.5	29.7
17	1747	–	0.1860				
	1837	0.3976	–	10	15	23.5	28.8
18	1904	–	0.3407				
	2008	0.3574	–	5	10	21.4	25.8
19	2098	–	0.4161				
	2181	0.3238	–	4	8	20.3	25.2
20	2297	–	0.2991				
	2305	0.3215	–	4	8	20.0	25.0

Table 5: Evolution of a typical 2CCA network trained on S^{100}_{train}.

A Appendix

This appendix contains all the data from the two typical runs described in Section 5.3. Table 4 and Table 5 show the most important figures and the percentages of misclassified training and test patterns for the HCCA and the 2CCA network, respectively.

"# hid." is the number of hidden units installed, "EI" is the ErrorIndex, and "Corr." is the magnitude of the correlation between the hidden units installed and the residual error of the primary network. "Euclid" ("Bin") refers to the Euclidean (Binary) method, respectively, of measuring the failure rate. The first line in the i'th row tells when the i'th hidden unit was installed and the magnitude of its correlation with the residual error of the primary network. The second line shows the performance of the network after the unit has been installed and all the output weights trained. These are the data that underlie the graphs in Section 5.3.

Evolution of a typical HCCA network							
# hid.	N_{epoch}	EI	Corr.	E^{100}_{train}(Euclid)	E^{100}_{train}(Bin)	E^{100}_{test}(Euclid)	E^{100}_{test}(Bin)
	37	0.9262	–	54	76	59.0	79.4
1	99	–	0.7630				
	138	0.8324	–	47	59	53.0	63.5
2	207	–	0.7025				
	257	0.7555	–	39	60	50.0	64.9
3	305	–	0.5258				
	347	0.7042	–	36	46	44.5	57.5
4	424	–	0.7935				
	450	0.6711	–	30	40	41.7	53.7
5	488	–	0.5304				
	543	0.6324	–	24	39	41.8	53.8
6	609	–	0.7029				
	651	0.5852	–	19	28	38.5	44.1
7	727	–	0.5526				
	761	0.5462	–	18	24	38.4	42.4
8	843	–	0.6989				
	887	0.5027	–	17	21	36.8	41.0
9	967	–	0.7963				
	1029	0.4688	–	16	20	36.4	40.9
10	1082	–	0.5380				
	1122	0.4425	–	15	19	37.5	42.1
11	1199	–	0.6042				
	1273	0.3977	–	13	17	33.8	38.4
12	1372	–	0.6035				
	1448	0.3445	–	9	16	32.3	38.5
13	1535	–	0.9985				
	1653	0.2668	–	3	6	30.8	34.4
14	1746	–	2.0704				
	1770	0.1999	–	1	2	30.2	32.5

Table 4: Evolution of a typical HCCA network trained on S^{100}_{train}.

[Mitchell80] Mitchell, T. M.
The Need for Biases in Learning Generalizations
Technical Report CBM-TR-117, Dept. of Comp. Sci., Rutgers University, New Brunswick, NJ, 1980.

[Mitchell82] Mitchell, T. M.
'Generalization as Search'
In *Artificial Intelligence*, Vol. 18, pages 203–226, 1982.

[PDP] Rumelhart, David E., McClelland, James L., and the PDP Research Group
Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations
Cambridge: MIT Press 1988.

[Rosenblatt] Rosenblatt, F.
Principles of Neurodynamics
New York: Spartan Press, 1962.

[Rosenberg] Rosenberg, C. R.
'Revealing the structure of NETtalk's internal representations'
In *Proceedings of the Ninth Annual Conference of the Cognitive Society*, 1987.

[Sejnowski *et al.*] Sejnowski, T. J., and Rosenberg, C. R.
'Parallel networks that learn to pronounce English text'
In *Complex Systems*, 1:145-168, 1987.

[Sjøgaard] Sjøgaard, Steen
Fingerprint Recognition and Discrete Neural Networks
Skriftligt projekt, Aarhus Universitet, Datalogisk Afdeling 1991. (In preparation)

[Widrow *et al.*] Widrow, B., and Lehr, M. A.
'30 Years of Adaptive Neural Networks: Perceptron, Madelaine, and Backpropagation'
In *Proceedings of the IEEE*, Vol. 78, No. 9, September 1990.

[Wieland *et al.*] Wieland, A. and Leighton, R.
'Geometric analysis of neural network capabilities'
In *Proc. 1st Int. Conf. on Neural Nets*, IEEE, San Diego, CA, June 1987, Vol. 2.

[Hornik *et al.*] Hornik, K., Stinchcombe, M., and White, H.
'Multi-layer feed-forward networks are universal approximators'
In *Neural Networks*, No. 2, 1989.

[Huang/Huang] Huang, Shih-Chi and Huang, Yih-Fang
'Bounds on the Number of Hidden Neurons in Multilayer Perceptrons'
In *IEEE Transactions on Neural Networks*, Vol. 2, No. 1, January 1991.

[Huang *et al.*] Huang, W. H., and Lippmann, R. P.
'Neural net and traditional classifiers'
In *Proc. Conf. on Neural Information Processing Systems*, Denver, CO, November 1987.

[Ilsøe *et al.*] Ilsøe, Lars and Poulsen, Ulrik
Introduktion til lagdelte neurale net og vækstalgoritmer
Master thesis, Datalogisk Institut, Københavns Universitet, 1989.

[Jakobsen *et al.*] Jakobsen, Jacob and Jensen, Tommy Ravn
Generalisering i Neurale Netværk
Skriftligt projekt, Aarhus Universitet, Datalogisk Afdeling 1990.

[Krogh] Krogh, Anders
Learning and Generalization in Neural Networks
Ph.D thesis, The Niels Bohr Institute, University of Copenhagen, 1991.

[Lippmann] Lippmann, R. P.
'Computing with neural nets'
In *IEEE ASSP Magizine*, Vol. 4, No. 2, April 1987.

[Lorentz] Lorentz, G. G.
'The 13'th problem of hilbert'
In *Mathematical Developments Arising from Hilbert Problems*, editor F. E. Browder, American Mathematical Society, Providence, Rhode Island, 1976.

[McClelland] McClelland, James L., and Rumelhart, David E.
Explorations in Parallel Distributed Processing. A handbook of models, programs, and exercises
Cambridge: MIT Press 1988.

[Mézard *et al.*] Mézard, M., and Nadel, J.-P.
'Learning in Feedforward Layered Networks: The Tiling Algorithm'
In *Journal of physics A 22*, page 2191-2204, 1989.

[Minsky/Papert] Minsky, M. and Papert, S.
perceptrons
Cambridge, MA: MIT Press, 1969.

References

[DARPA] *DARPA Neural Network Study. October 1987 – February 1988*
AFCEA International Press.

[Fahlman] Fahlman, Scott E.
'Faster-Learning Variations on Backpropagation: An Empirical study'
In *Proceedings of 1988 Connectionist Model Summer School*, Editors Touretzky, D., Hinton, G., and Sejnowski T., San Mateo, CA, Morgan Kaufmann, pp. 38-51.

[Fahlman *et al.*] Fahlman, Scott E., and Lebiere, Christian
The Cascade-Correlation Learning Architecture
Technical Report CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213.
Also in *Advances in Neural Information Processing Systems II*, editor David S. Touretzky, pages 524–532. San Mateo: Morgan Kaufmann Publishers.

[Frean] Frean, M.
'The Upstart Algorithm: A Method for Constructing and Training Feed-forward Neural Networks'
In *Neural Computation 2*, page 198-209, 1990.

[Gibson/Cowan] Gibson, Gavin J. and Cowan, Colin F. N.
'On the Decision Regions of Multilayer Perceptrons'
In *Proceedings of the IEEE*, Vol. 78, No. 10, October 1990.

[Hao *et al.*] Hao, J., Tan, S., and Vandewalle, J.
'A Geometric Approach to the Structural Synthesis of Multilayer Perceptron Neural Networks'
In *INNC*, Vol. 2, July 9–13, 1990, Palais des Congres, Paris, France.

[Haussler] Haussler, D.
Quantifying Inductive Bias in Concept Learning
Technical Report UCSC-CRL-86-25, Dept. of Comp. Sci., University of California, Santa Cruz, CA, 1986.

[Hertz *et al.*] Hertz, J., Krogh, A., and Palmer, R. G.
Introduction to the theory of neural computation
Lecture notes Volume I, Santa Fe Institute, Addison-Wesley, 1991.

[Honavar *et al.*] Honavar, Vasant, and Uhr, Leonard
'Generation, Local Receptive Fields and Global Convergence Improve Perceptual Learning in Connectionist Networks'
In *Parallel and Distributed Processing*, IJCAI-89, editor N. S. Sridharan, San Mateo, Morgan Kaufmann Publishers, Inc., CA.

networks undergo during the training process. It is our hope that this thesis has provided the reader the same benefit.

weights are frozen. Now, this procedure implies that one of two things will happen:

1. If the winning unit came from the side pool, then the number of units in the expansion layer will be increased by one.

2. If it came from the front pool, the network will be expanded with a new layer consisting of the winning unit. That layer then becomes the new expansion layer (i.e. layer $k-1$) while the size of the previous expansion layer (i.e. layer $k-2$) will be fixated.

This expansion scheme is only a little more complicated than the schemes employed by the HCCA and the 2CCA algorithms. There is, however, a small irregularity that should be noticed, namely that the expansion layer always should be "expanded" by at least one unit before the next layer of hidden units is introduced. The reason for this is obvious, since otherwise it would be possible to commence a new layer of units which are connected to only a single unit in the previous layer. That is, only the side pool should be brought into action immediately after a new layer of hidden units has been introduced.

This algorithm possesses the two major advantages of the HCCA and the 2CCA algorithms: The networks it creates are capable of computing higher-order correlations, and the size or dimension of their hypothesis spaces will only grow linearly with the number of hidden units installed. So, even though the networks are capable of dealing with very powerful hypothesis representations, there is reason to believe that the version spaces will not be oversized, and thus that the possibility of misclassification of future events will be poor.

In continuation of the careful studies and comparisons of the two cascade correlation learning algorithms that have been presented in the latter half of this thesis, we find the k-CCA algorithm briefly described above very interesting. At least on the paper, it possesses a large number of qualities which the preceding analyses have proven to be of great value when the main topic of interest is generalization. If more time had been available for this project, it would have been very interesting to go into a thorough analysis of its performance in practice.

Final Remarks

The main objective of this thesis was to propose a conceptual approach to generalization in artificial neural networks. To meet this objective, we have introduced a conceptual framework which is independent of specific learning and generalization strategies, and which therefore provides a terminology that render possible to compare and discuss alternative classes of learning algorithms and to describe the different representations for learned generalizations they employ. Furthermore, we find that the two basic ideas of (1) viewing the problem of generalization as a search problem, and (2) to divide the search space into a static and a dynamic part (i.e. the introduction of the version space concept) are intuitively appealing. In any case, the approach has provided the author an improved comprehension of the process of generalization and the evolution that artificial neural

Discussion and Possible Improvements

In the light of the discussions throughout the thesis and the obtained results, we conclude that the 2CCA algorithm is preferable to the HCCA algorithm in many ways. Especially when we focus on the generalization ability, all the 2CCA networks have proven to be superior to the HCCA networks. And the encouraging results obtained in Experiment 2 have emphasized this further, since, by allowing three types of hidden units, it is possible to reduce the number training epochs by a third and at the same time remain the fine generalization ability. However, even though the analyses and the experiments carried out in this thesis clearly have shown the superiority of the 2CCA algorithm, one should be aware of the limitations underlying the encouraging results. For example, the algorithm has only been tested on a relatively simple geometric classification problem. Its performance on a more realistic and large-scaled problem is still unknown. Furthermore, we have only compared it with the HCCA algorithm which is very similar. A comparison with a static learning algorithm as e.g. backpropagation (it seems to be the standard algorithm to which all other algorithms are compared) and, of course, quickprop is obvious, and if more time had been available for this work, the thesis would of course have included such comparisons.

A k-CCA algorithm

As mentioned above, the 2CCA algorithm has proven to be superior to the HCCA algorithm in many ways. However, as we mentioned on page 42, the two-spirals problem (as Fahlman and Lebiere chose as the primary benchmark for their study) seems to be very hard for the 2CCA algorithm to solve, whereas the HCCA algorithm performs very well on the problem. This indicates that multi-layer networks, due to their capability of computing higher-order correlations, still have some benefits that the 2CCA networks do not possess. A symbiosis of the two algorithms which does not expand the version space no more than necessary and which is capable of computing higher-order correlations, might therefore be advantageous. However, such a k-layer cascade-correlation learning algorithm (k-CCA) involves a new problem, namely how to decide at which layer a new hidden unit should be installed. The most obvious way to tackle this problem is to define that all hidden layers but layer $k-1$ should be fixed, i.e. only layer $k-1$ (the hidden layer immediately before the output layer) may be expanded. Let us call that layer the *expansion layer*. Then there are only two possibilities when the primary network need to be expanded: Should a new hidden unit be added to the expansion layer, or should yet another layer of hidden units be introduced. One way to tackle this problem is to operate with two pools of candidate units simultaneously, namely a *front pool* and a *side pool*. The candidates in the front pool receive a connection from all the units in the expansion layer (i.e. from the units in layer in layer $k-1$), while the candidates in the side pool receive a connection from all the units in the immediately preceding layer, i.e. from the units in layer $k-2$. And as usual, when these candidate units are being trained, all the weights in the primary network are temporarily frozen. This implies that the candidates do not interact with one another, and thus that both pools of candidates can be trained simultaneously. The unit from the two candidate pools whose output exhibits the maximum correlation with the residual error of the primary network is added and its input

to minimize the risk of misclassification of future events, the version space should be as small as possible. That is, the networks should not be able to deal with hypotheses that are more complex than necessary, and therefore the network architectures should be as simple as possible. Once again our attention was directed to the problems of network design, and the benefits of applying dynamic learning algorithms were stressed again.

In order to demonstrate the importance of the hypothesis representation and its impact on the generalization ability, we thoroughly analyzed two specific dynamic learning algorithms. The first algorithm (HCCA) was developed by Fahlman and Lebiere [Fahlman *et al.*], while the other one (2CCA) was developed by the author as a suggestion for a dynamic learning algorithm that possesses the properties described above. In Chapter 4 and 5, these two algorithms were analyzed theoretically and empirically. The theoretical considerations in Chapter 4 made us put forward two assertions about the capabilities of the two learning algorithms: Due to the capability of the HCCA algorithm to form very powerful hypothesis representations, we expected (1) that these networks would be able to learn the set of training examples much quicker (in number of epochs) than the 2CCA networks, and (2) that their generalization ability would be poorer than for the 2CCA networks. Both of these assertions were empirically confirmed by the two experiments we carried out. The first experiment concerned the scaling behavior of the two algorithms. We found that the HCCA networks learned the set of training examples approximately twice as fast (in number of epochs) than the 2CCA networks, and that the 2CCA networks performed approximately 7% – 8% better than the HCCA networks on the test set. Furthermore, it was demonstrated how the conceptual framework that we have introduced made possible to explain the increasing variation in the quality of the generalizations discovered by the HCCA networks. Their version space simply became too large when the size of the training set exceeded 500 examples. This adverse property did not appear for the 2CCA networks.

In the second experiment, we analyzed the importance of the hypothesis representation more thoroughly. By allowing different types of hidden units (characterized by their output function), it was demonstrated how the learnability of the two network types was improved. The number of training epochs decreased by approximately a fourth for the HCCA networks, while the corresponding reduction for the 2CCA networks was approximately a third. The generalization ability of the HCCA networks, on the other hand, was deteriorated by allowing the three types of hidden units. Again we demonstrated how the version space concept rendered possible to explain this result: We observed that the variation in the quality of the hypotheses discovered increased from 3.8% to 9.9%, so the networks' possibility of dealing with more powerful hypothesis representations implied that the size of their version space increased, and thereby the possibility of misclassification as well. For the 2CCA networks, the variation in the quality of the hypotheses discovered remained at approximately 2%, and, exactly as expected, the generalization ability therefore remained at the same level too. Actually we saw a slightly improvement of the generalization ability from 89.4% to 90.4%.

6 Conclusion

In the beginning of this thesis the capabilities of a special class of feed-forward networks, called two-layer Perceptrons, were discussed thoroughly from a geometric point of view. First, we demonstrated that this class of relatively simple networks is somewhat more versatile than was originally believed [Lippmann], [DARPA]. We showed that the decision region of a two-layer Perceptron is a union of a finite number of convex sets, and in terms of hypercube vertices and hyperplane cuts, we stated the necessary and sufficient conditions for a region in $\mathbb{R}^n$ to be realized as a decision region for a two-layer Perceptron. Then we directed our attention to the problems of network design. But unfortunately we found that the necessary number of hidden units depends on topological properties of the function that the Perceptron is intended to realize. Instead we therefore focussed on the learnability of two-layer Perceptrons, but again the results were disappointing: It appeared that the least upper bound on the number of hidden units for a two-layer Perceptron to be capable of realizing arbitrary functions defined on a k-element set, is $k-1$. Moreover, we discussed the V-C dimension of a class of functions to be realized by two-layer Perceptron, but again the results were much too theoretical to be of any practical use. These disappointing theoretical results emphasized the serious problems of network design and function realization. As a suggestion for how most of these problems can be overcomed in practice, we argued for the benefits of employing dynamic learning algorithms.

Inspired by the work of Tom Mitchell [Mitchell82], we proposed in Chapter 3 a conceptual approach to generalization in artificial neural networks. After a survey of Mitchell's work, it was demonstrated how the conceptual framework can be applied to artificial neural networks. By considering the overall error function as an inverse utility function for the different generalizations — or hypotheses — discovered during the training process, we obtained a total ordering of all possible hypotheses. We argued that the utility concept is less strict than the corresponding consistency concept applied by Mitchell in the context of Machine Learning. Then we used it to define the version space concept, and we demonstrated how the looseness of the utility concept actually became the strength of the conceptual framework in the context of artificial neural networks. The utility concept namely guarantees (when a gradient descent learning algorithm is applied) that the size of the version space will be reduced after each weight update, and that more and more training instances automatically are correctly classified by the sequence of current best hypotheses discovered. That is to say that the consistency of the final hypothesis is ensured automatically! However, at this point a very important question should be addressed, namely whether this final hypothesis also guarantees a satisfactory generalization ability, i.e. whether the hypothesis enables the network to correctly classify future events. This issue became subject for most of the discussions in the rest of the thesis.

In continuation of Mitchell's definition of "bias," we suggested a distinction between four types of different biases that appear in artificial neural networks. Especially the importance of the hypothesis representation was discussed thoroughly, and in short we found that the more powerful the representation is, the larger is the version space, and consequently the possibility of misclassification of future events will increase. This result emphasized further the benefits and the usefulness of the version space concept: In order

2CCA networks. And now that we have demonstrated how the number of training epochs can be reduced by a third, the superiority of the 2CCA algorithm is emphasized further.

The next thing we notice is that the dimension of the hypothesis space is reduced from dimension 306 to dimension 191, i.e. a 38% reduction which is considerable more than we obtained by the HCCA networks. Actually the average hypothesis space for a 2CCA network is only half as large as the hypothesis space for a HCCA network (dimension 191 and dimension 374, respectively). And as we can see in the last two rows in Table 3, the generalization ability of the 2CCA networks is retained. This means that we actually have achieved an improved learnability without paying the cost of a deterioration of the generalization ability! And as we can see in the last row, the variety of the consistent generalizations discovered remain unaltered as well, which indicates that the quality of the generalizations in the version space is practically unaffected by the increased complexity of the hypothesis representation that the networks are enabled to deal with. That is to say that the deterioration of the generalization ability due to the enhanced degree of freedom of the hidden units is counterbalanced by the smaller version space. Out of the ten runs with optionality in the selection of output function, the *poorest* generalization discovered resulted in 89.2% correctly classified test patterns while the *best* generalization gave 91.5% correctly classified test patterns. The corresponding percentages for the ten runs with the asymmetric sigmoid output function were 87.9% and 90.1%, respectively. (See Appendix B and C.) So, with a bit of good will, we can actually see a small tendency to an *improved* generalization ability for the 2CCA networks, when the three output functions are available for their hidden units.

Now, let us try to summarize the obtained results when the complexity of the hypothesis representation is increased. First of all, we have obtained an improved learnability for both network types by introducing the optionality in the selection of output function for the hidden units. For the HCCA networks, the necessary number of training epochs is reduced by a fourth (from 2999 to 2252 epochs), and for the 2CCA networks it is reduced by a third (from 5601 to 3627). Likewise, the number of hidden units installed decreased, and this has caused that the dimension of the hypothesis space of an average HCCA network is reduced by 24% (to dimension 374), while the dimension of the hypothesis space a HCCA network is reduced by 38% to dimension 191. Thus, the largest improvement of the learnability is achieved with the 2CCA learning algorithm.

The 2CCA learning algorithm also displays the best results when we focus on the trained networks' generalization ability: The 2CCA networks have retained (or perhaps even slightly improved) their performance on the test set S_{test}^{1000} (90.4% of the test patterns are correctly classified), while the rate of correctly classified test patterns by the HCCA networks has decreased from 83.1% to 78.2%. We also notice that the quality of the generalizations discovered by the 2CCA networks is unaffected by the improved learnability, while the variation in the quality of the generalizations discovered by the HCCA networks increased from 3.8% to 9.9%.

We can therefore conclude that there are absolutely no drawbacks attached to the 2CCA algorithm by using the composition of output functions described in this section for the hidden units, in order to increase the complexity of the networks' hypothesis representation. For the HCCA algorithm, on the other hand, the cost of using the three different output functions is a deterioration of the networks' generalization ability. This result supports our discussions in the previous sections about the nice properties of the

moid family. We will not examine this relation any further in this thesis, but we find the result very interesting and further investigations into the area of variable output functions are obvious. Indeed this experiment shows that the choice of output function is of crucial importance, and it indicates that in general a more thorough analysis of the artificial neuron as the fundamental building block of artificial neural networks is required. For the interpretation of the results obtained in this experiment we shall therefore only conclude that the extra optionality of the algorithms has turned out to be beneficial for the learnability.

The smaller number of hidden units has reduced the dimension of the hypothesis space by 24% from dimension 492 to dimension 374. This means that the size of the version space has been reduced considerably as well, but as we can see in Table 3, the number of correctly classified test patterns from S_{test}^{1000} has declined from 83.1% to 78.2%. That is, the generalization ability has been deteriorated in spite of the smaller version space. The point is, however, that the smaller version space actually contains a broader variety of consistent generalizations. This is easily seen in the last row in Table 3 which shows the range in performance between the best and the poorest generalization found: When the algorithm is allowed to operate with the three different output functions for the hidden units, the variation of the quality of the generalizations discovered is much larger (9.9%) than when only the asymmetric sigmoid function is allowed (3.8%). And as in the previous experiment, we interpret this as an indication of an oversized version space. Out of the ten runs with optionality in the selection of output function, the *poorest* generalization discovered resulted in only 73.6% correctly classified test patterns while the *best* generalization gave 83.5% correctly classified test patterns. The corresponding percentages for the ten runs with the asymmetric sigmoid output function were 80.8% and 84.6%, respectively (see Appendix B and C). So indeed the similarity of the generalizations discovered when just the asymmetric sigmoid output function is allowed is much greater than when the three different output functions are available. The conclusion is therefore that it is not only the dimension of the hypothesis space (i.e. the number of weights or free parameters) that are decisive for a network's generalization ability. The outcome of this experiment has clearly substantiated our considerations in the previous sections, namely that the generalization ability is affected by the complexity of the network's hypothesis representation as well. We will discuss this issue more thoroughly in a moment, but first we want to focus on the results obtained by the 2CCA networks.

The 2CCA networks: Also for the 2CCA networks the learnability is improved compared to the corresponding results obtained in Experiment 1: The number of training epochs is reduced by 35% and the number of hidden units by 38%. So, the effect on the learnability by using the three different output functions is actually larger for the 2CCA networks than for the HCCA networks.

If we focus on the type of hidden units that have been installed, we notice that the rate of Gaussian functions is 57% while approximately a third of the units now use the asymmetric sigmoid output function. Again it is almost impossible to explain this distribution, but we notice that all three types of functions are more evenly represented than they were in the HCCA networks, and we conclude that the extra optionality of the algorithm has improved the learnability considerably.

Improving the Learnability						
	HCCA			2CCA		
	fixed *out*	mixed *out*	Δ	fixed *out*	mixed *out*	Δ
asigm	100%	11%	–	100%	35%	–
ssigm	–	5%	–	–	8%	–
gauss	–	84%	–	–	57%	–
N_{epoch}	2999	2252	↘ 25%	5601	3627	↘ 35%
N_{hid}	26.9	22.9	↘ 15%	60.4	37.3	↘ 38%
N_{con}	492	374	↘ 24%	306	191	↘ 38%
Performance	83.1%	78.2%	↘ 4.9%	89.4%	90.4%	↗ 1%
Range	3.8%	9.9%	↗ 6.1%	2.2%	2.3%	– 0%

Table 3: A comparison of the averaged results obtained by the two network types when only the asymmetric output function is applied (the "fixed *out*" columns) and when the three different output functions are allowed (the "mixed *out*" columns) for the hidden units. The first three rows show the distribution of the output functions in the two network types. See text for further explanation.

for the experiments as we find its size and the results obtained with it in Experiment 1 appropriate. And, as usual, each experiment is carried out ten times with different random weights.

A comparison of the average of the most important results obtained for the two network types is shown in Table 3. The figures in the two "fixed *out*" columns are copied from the results presented in Table 2 and refers to the use of the asymmetric sigmoid function as output function for all the hidden units. The results in the two "mixed *out*" columns are obtained by allowing the networks to operate with the three output functions described above, and in the "Δ" columns the differences between the two experiments are shown. The exact figures for all the experiments can be found in Appendix B and C.

As before, let us examine the obtained results for the two network types separately.

The HCCA networks: First of all, we notice that the learnability indeed is improved compared to the corresponding results obtained in Experiment 1. This is easily seen in the "Δ" column (a '↘' ('↗') indicates a decrease (increase), respectively), and we observe that the number of training epochs is reduced with 25%, while the number of hidden units is reduced with 15%. This indicates that the number of epochs used to train and select the candidate units has decreased, i.e. the hidden units do not only eliminate a larger part of the residual error, but they also eliminate that part somewhat quicker.

If we focus on the type of hidden units that have been installed, we see that 84% of the units use the Gaussian output function, while only 5% (11%) use the symmetric (asymmetric) sigmoid output function, respectively. Solely from the results presented above it is almost impossible to explain this distribution. We can state, however, that on this particular geometric classification problem, units using the Gaussian output function are more likely to produce outputs that exhibit a larger correlation with the residual error than units using output functions from the sig-

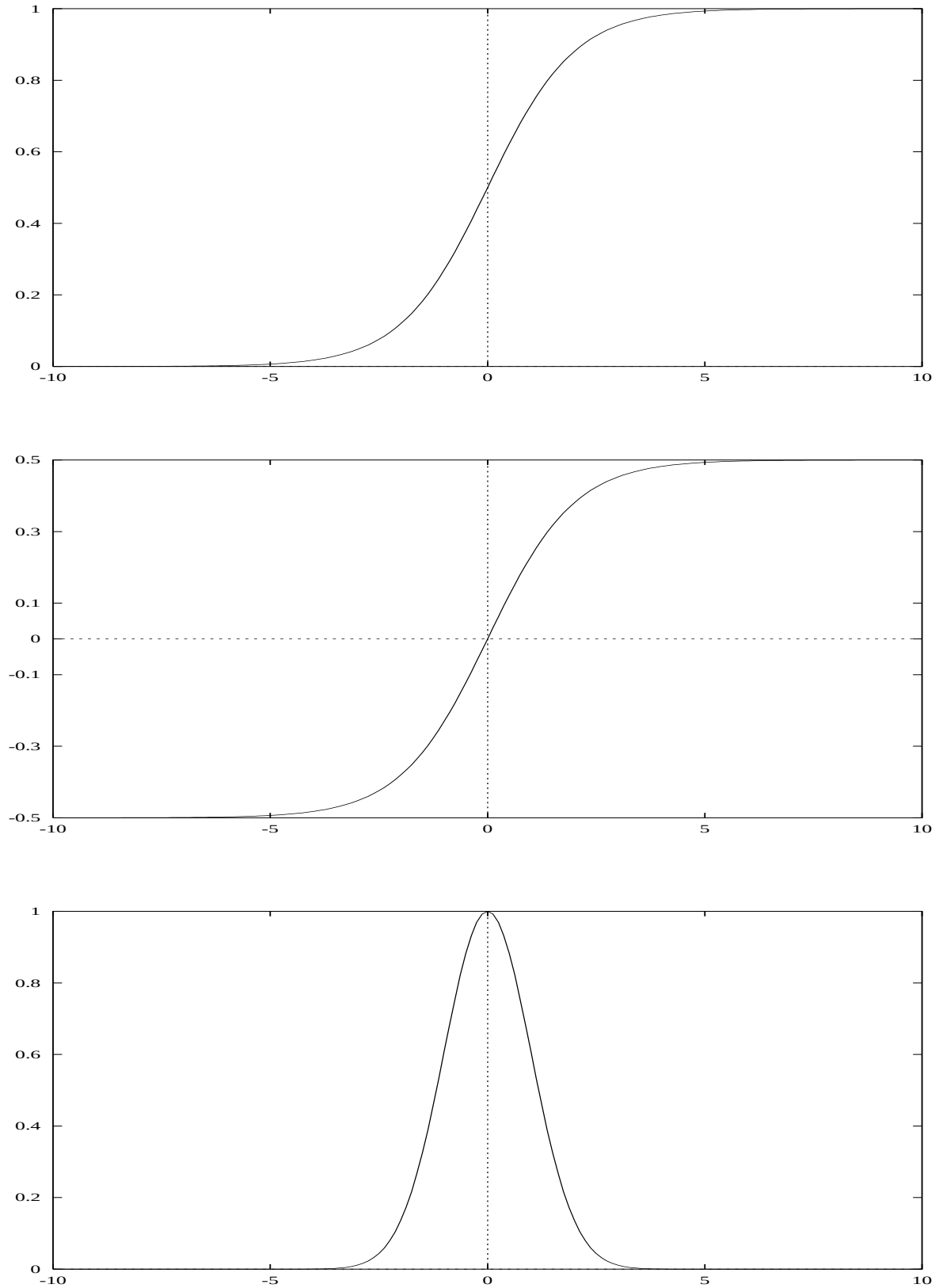

Figure 25: The graphs of the three output functions applied in experiment 2. The upper graph is *asigm*, the one in the middle is *ssigm*, and the lowest is *gauss*.

Finally, the third type of units use the Gaussian function *gauss*:

$$out(a) = gauss(a) = \exp(-0.5\,a^2)$$

The *gauss* function is characterized by a relatively small symmetric region where its output is high. This means that hidden units using this output function not will contribute to the overall computation if their input is numerical high. Indeed such units will be specialized as their active region is rather cramped.

The graphs of the three functions are shown in Figure 25. The candidate pools generated will always consist of three units from each of these three types, i.e. they will contain three asymmetric sigmoid units, three symmetric sigmoid units, and three Gaussian units. This increase the chance of the creation of feature detectors that will exhibit a much better fit with the network's overall computation and hence that the residual error of the network will be reduced much faster. That is to say that the hidden units now can evolve into feature detectors that are even more specialized then before, and therefore we expect that much fewer hidden units need to be installed, and hence that the number of training epochs will be reduced considerably. All in all the learnability of the networks should therefore be improved by the increased variety of the hidden units.

When we focus on the networks' generalization ability, on the other hand, the situation is more complicated. According to our discussions in the two previous chapters, we can expect that the networks' generalization ability will be deteriorated due to the enhanced degree of freedom of the hidden units. However, the reduced number of hidden units in the final networks due to the improved learnability, imply that the size of the hypothesis spaces will be reduced as well. And in Experiment 1 above we saw that a small hypothesis space, and hence a small version space, is beneficial for the networks' ability to generalize. There is therefore a kind of tradeoff here: On the one hand, the enhanced degree of freedom of the hidden units will deteriorate the generalization ability, while, on the other hand, the fewer number of hidden units needed in the final networks will cause a smaller hypothesis space, which is beneficial for the generalization ability. The total effect of increasing the "amount of unrestrictedness" by allowing the learning algorithms to operate with three different types of output functions for the hidden units to be installed as highly specialized permanent feature detectors, is therefore difficult to predict. We believe, however, that if the hypothesis space is reduced sufficiently, we *might* achieve that the networks' generalization ability is retained, and this is the main reason why we want to carry out this experiment. Maybe we can obtain an improved learnability *without* having to pay the cost of a deterioration of the generalization ability. It is, however, more likely that the extra "amount of restrictedness" resulting from the reduced hypothesis space not will be in proportion to the increase of the "amount of unrestrictedness," and hence we will see a reduced generalization ability compared with the results obtained in Experiment 1.

All the parameter settings for the two algorithms are identical to the settings applied in Experiment 1 (see Table 1 on page 45), except for 'ncandidates' which is changed from 8 to 9, and 'unittype' which is changed from 'ASYMSIGMOID' to 'MIX3'. This means that each time a new hidden unit has to be installed, the algorithms generate three candidates with the asymmetric sigmoid output function, tree candidates with the symmetric sigmoid output function, and three candidates with the Gaussian output function which then comprise the candidate pool. Furthermore we have decided to use the training set S^{500}_{train}

of the generalizations found by the 2CCA networks steadily decreases when the number of training examples is increased. This is a property that is in accordance with what one intuitively expects, and it indicate that the repeatedly expansion of the version space proceeds at a pace which is appropriate to encourage similarity of the generalizations discovered. The version space is expanded no more than necessary.

For the HCCA networks, on the other hand, we saw that the variation in the quality of the generalizations discovered increases when the number of training examples exceeds 500. This is a consequence of an oversized version space, and we saw that an increase from 500 to 1000 training examples not necessarily guarantees an improved generalization ability for these networks. So, even though the HCCA networks learns the training set very fast (in number of epochs), several runs might be necessary in order to obtain an improved generalization ability. This is an awkward property which together with the in no way impressive performance in classifying novel examples, lead us to conclude that the 2CCA learning algorithm in many ways is preferable to the HCCA algorithm developed by Fahlman and Lebiere, at least on the relatively simple geometric classification problem we consider in this thesis.

5.5 Experiment 2: Improving the Learnability

In this section we will study the importance of the hypothesis representation more thoroughly. What we are interested in, is to see what impact an improved learnability has on the networks' generalization ability: In Experiment 1 above, all the hidden units that were installed, were selected from a pool of candidate units that were trained simultaneously and which only differed in the initial random weights. The candidate unit that exhibited the maximum correlation with the residual error of the primary network was selected and installed as a permanent feature detector. In the experiments carried out in this section, we shall increase the variety of the units in the candidate pool by generating three different types of candidate units characterized by the output function they use. The first type of units use the same output function as in Experiment 1, namely the asymmetric sigmoid function $asigm$:

$$out(a) = asigm(a) = \frac{1}{1 + \exp(-a)}$$

The second type of units use the symmetric sigmoid function $ssigm$:

$$out(a) = ssigm(a) = \frac{1}{1 + \exp(-a)} - 0.5$$

$ssigm$ is identical to the $asigm$ function, except that all output values are translated by -0.5, which imply that both positive and negative output values will occur. This has an effect on the sign of the correlations computed: Suppose that two different training patterns are dual in the sense that they demand approximately the same network configuration, except for the sign of the weights, in order to be correctly classified. If the $ssigm$ output function is used, then both of these can be correctly classified whereas one of them would be misclassified if the $asigm$ output function was used.

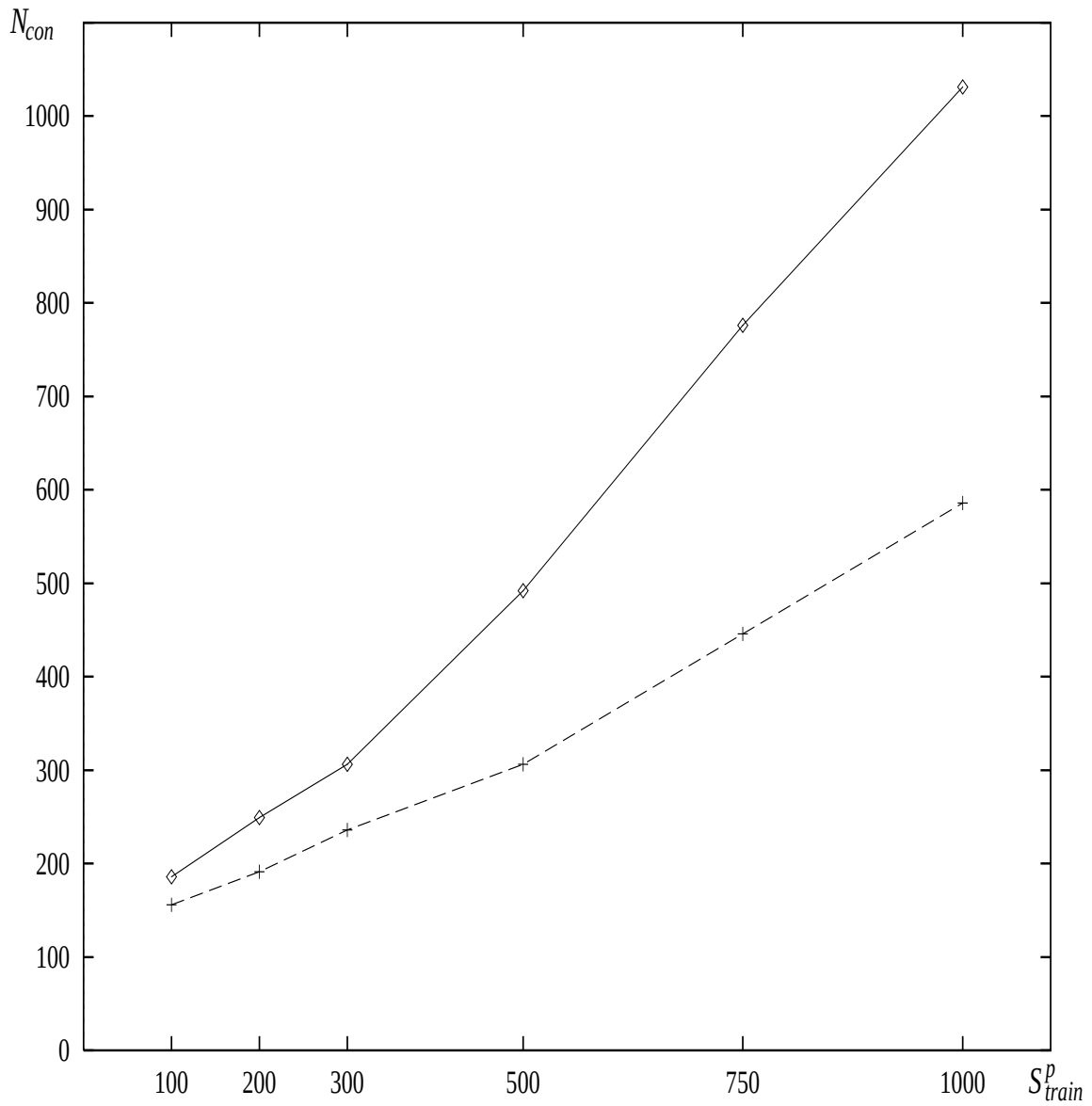

Figure 24: The number of connections in the final networks a function of the number of examples in the six training sets. The diamonds ($\diamond$) represent the HCCA networks, while the plusses ($+$) represent the 2CCA networks.

sets, we may in fact see in Figure 22 that the best network trained on S^{500}_{train} generalizes *better*[23] than the poorest network trained on S^{1000}_{train}. This and the growing tendency to a larger variation in the quality of the generalizations discovered is a plain indication of an oversized version space for these networks! And if we assume that the domain bias is negligible since p is relatively large, this can only be caused by the initial random weights (i.e. by algorithmic biases, ref. Section 3.2) and by the different networks' hypothesis representation.

Since we focus on the averaged performances, we permit ourselves to temporarily ignore the algorithmic biases, and thus the only explanation of this developmental tendency of the networks must be found in their ability to represent progressively more complex hypotheses. To substantiate this assertion, we have depicted the averaged dimension of the hypothesis spaces (i.e. N_{con}) as function of the training set size in Figure 24. And now it is easy to see how fast the number of free parameters in these networks increases when the number of constraints imposed by the training instances is increased. If we assume that the relation is linear when $p \geq 500$ (the graphs are very close to straight lines) we see that N_{con} for the HCCA networks increases approximately twice as fast as it does for the 2CCA networks. Furthermore we notice that the number of free parameters in these networks almost is equivalent to the number of training instances in the applied training sets, and if we recall our discussion in Section 3.2 on generalization in neural networks in general, we know that this means that some of the weights in the networks not will be appropriately constrained. That is, some of the input weights to the latest installed hidden units are not strictly necessary and thus they do not contribute with any important or essential information during the training of these units. This means that an increasing number[24] of unnecessary weights will contribute with more or less stochastic information due to effects of initial random conditions, and this, of course, will have a disturbing impact when these units are installed as permanent high-order feature detectors.

Now, let us summarize the discussions above and draw some final conclusions on the scaling behavior of the two network types. First of all, the expectations we proposed in Section 4.3.1 that the HCCA networks would be capable of learning the different training sets faster (in number of epochs) than the 2CCA networks, and that the 2CCA networks, on the other hand, would be superior to the HCCA networks at classifying the novel instances in the test set S^{1000}_{test}, have been confirmed by the experiments carried out in this section. In fact this relation between learnability and generalization ability for the two network types has shown to be independent of the training set size. However, Table 2 shows clearly that when the number of training examples exceeds 500, these two behavioral tendencies become even more conspicuous: The difference in number of training epochs for the two network types on S^{500}_{train}, S^{750}_{train}, and S^{1000}_{train} is 2602, 4124, and 5591, respectively, while the difference in performance on the test set S^{1000}_{test} is 6.3%, 7.8%, and 8.4%, respectively. Furthermore we have seen that the variation in the quality

[23]For a comparison, the performance of the best 2CCA network trained on S^{500}_{train} is 90.1% while the performance of the poorest 2CCA network trained on S^{1000}_{train} is 94.3%, a difference of 4.2% in the worst case.

[24]Recall that N_{con} according to Equation 5 on page 38 increases quadratically with the number of hidden units installed.

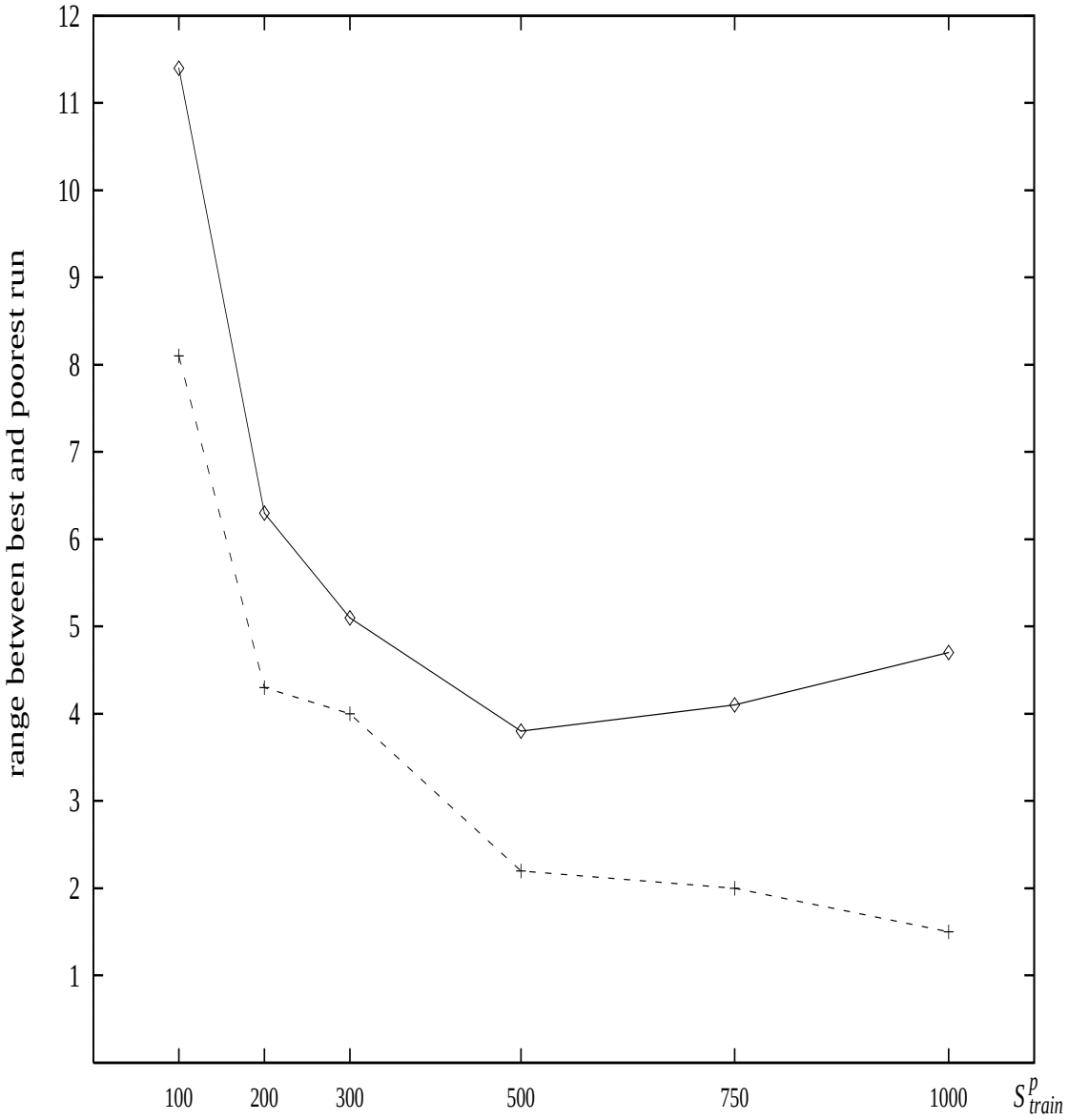

Figure 23: The range in performance between the best and the poorest run on the test set S^{1000}_{test} as function of the training set size. The diamonds ($\diamond$) represent the HCCA networks, while the plusses ($+$) represent the 2CCA networks.

using twice as many examples is considerable when p is this small.

However, more interesting is a direct comparison of the two curves, and first of all we observe that the 2CCA networks clearly outperforms the HCCA networks on all six training sets. It even looks like that the 2CCA networks derive a greater advantage than the HCCA networks from an increase of the number of training examples when $p \geq 500$, i.e. when the domain bias (from the probabilistic distribution of the training examples in the event space) is weakening. We shall analyze this issue more thoroughly below. Furthermore we notice that the range between the best and the poorest run on the first four training sets decreases with p. This development continuous steadily for the 2CCA networks, while it seems to reverse for the HCCA networks for p greater than 500. This is shown in Figure 23 which clearly illustrates these two different developmental patterns for the two network types.

To give a plausible explanation of this behavior, let us consider each of the two types of networks separately.

The 2CCA networks: The generalization ability of the 2CCA networks increases monotonically with p while the range between the best and the poorest run steadily decreases. This behavior is not very surprising if we recall that as p increases the version space gets smaller: In a *static* neural network, i.e. when the size of the hypothesis space is fixed, the number of hypotheses that are consistent with each of the p training examples will decrease when p is increased. (This relation is not totally strict. If the applied training set is expanded with m new examples and each of these m examples represent redundant information only, the situation will remain unaltered. However, the probability that at least one of the new examples will contradict some of the hypotheses from the set of currently consistent hypotheses, will increase with m.) The results in Figure 22 and 23 indicate that this result holds for the *dynamic* 2CCA networks as well even though their hypothesis space –and their version space– are expanded during the training process.

Seen in the light of our previous discussions on limitations and restrictions as a necessity to obtain a good generalization, we can think of this as if the "amount of restrictedness" imposed by increasing the number of training examples from 300 to 500 say, is greater than the "amount of unrestrictedness" resulting from the expansion of the version space from dimension 236 to dimension 306. (See Table 2.) We find that this is a sound property of these networks which is in accordance with what one intuitively expects of the behavior of neural networks in general.

The HCCA networks: The range between the best and the poorest run decreases monotonically for the first four training sets, while it slowly starts to increase for the two largest training sets S^{750}_{train} and S^{1000}_{train}. This development might at first sight seem a bit strange, but a plausible explanation can be found in continuation of the discussion above: When p is smaller than 500, the behavior of the HCCA networks is similar to the behavior of the 2CCA networks, i.e. the "amount of restrictedness" increases faster than the "amount of unrestrictedness." Then the development is reversed and we see a growing tendency to a larger variation in the quality of the generalizations discovered in different runs on the same training set. Furthermore, if we compare the performance of the networks trained on the three largest training

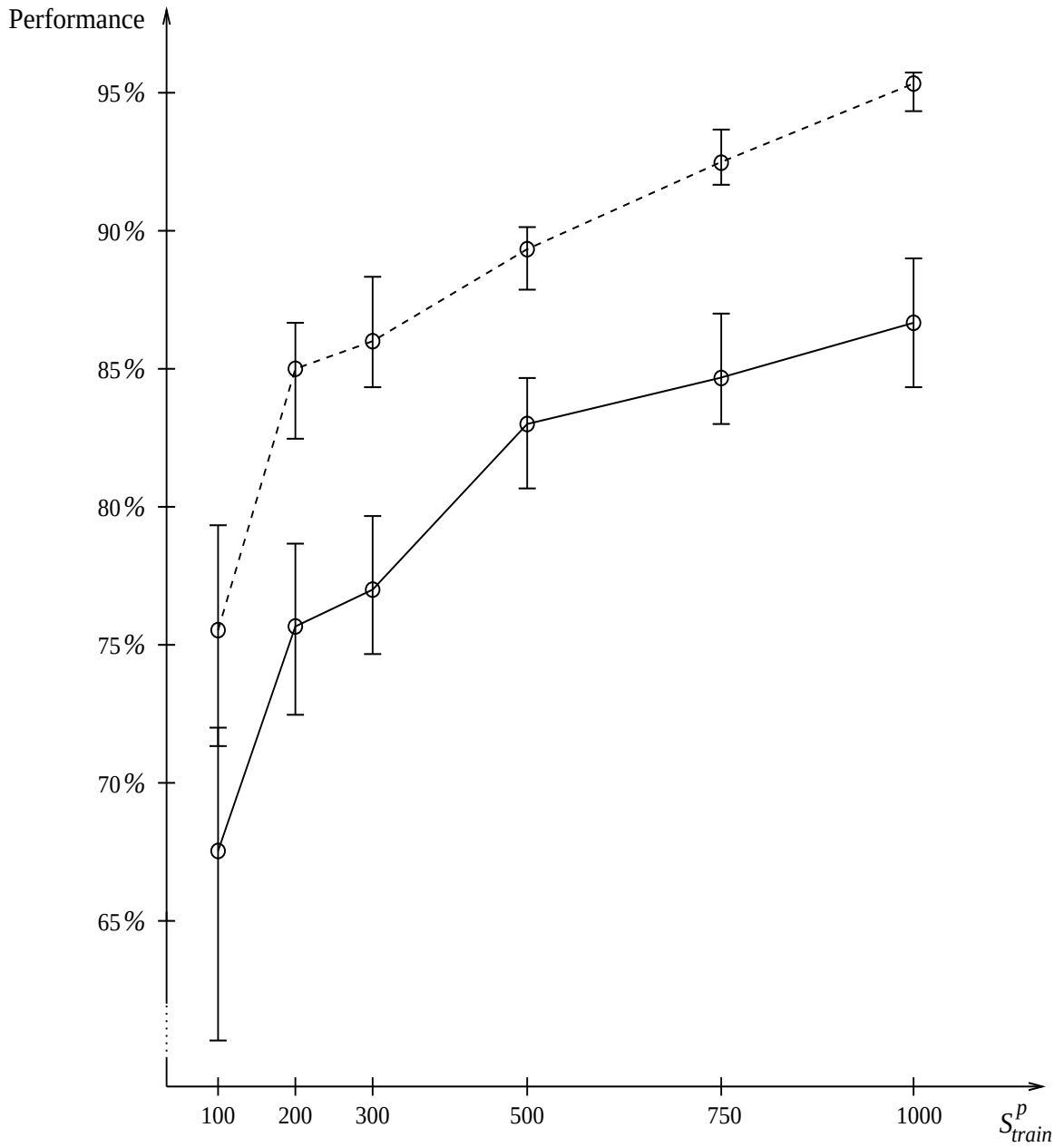

Figure 22: Performance of the final networks on the test set S^{1000}_{test} as function of the size of the six training sets. The dashed curve shows the performance of the 2CCA networks, while the HCCA networks are illustrated by the full-drawn curve.

Scaling Behavior								
	HCCA				2CCA			
training set	N_{epoch}	N_{hid}	N_{con}	perform.	N_{epoch}	N_{hid}	N_{con}	perform.
S_{train}^{100}	1752	14.5	186	67.5	3223	30.0	156	75.5
S_{train}^{200}	2133	18.4	249	75.8	3868	36.6	191	84.9
S_{train}^{300}	2666	23.9	306	77.1	4615	45.8	236	86.0
S_{train}^{500}	2999	26.9	492	83.1	5601	60.4	306	89.4
S_{train}^{750}	3606	34.5	776	84.8	7730	87.6	446	92.6
S_{train}^{1000}	3994	40.7	1031	86.8	9585	115.7	586	95.2

Table 2: Averaged results obtained by the two types of networks for each of the six training sets. N_{con} is the number of connections in the final networks (i.e. the dimension of the hypothesis spaces), and "perform." is the percentage of test patterns from S_{test}^{1000} that are correctly classified. See Figure 22 for a graphical presentation of the averaged performances.

That is, even though the HCCA network only has installed 14 hidden units, the dimension of its hypothesis space is approximately 10% larger than the hypothesis space of the 2CCA network which has installed 29 hidden units. And since we know that the HCCA network incrementally creates a more powerful hypothesis representation than the 2CCA network ever could (see Section 4.3), we see that the 2CCA network is subject to far more restrictions and limitations than the HCCA network. And that is precisely what we argued that would be beneficial for a network's ability to generalize. So, for the present it looks like that the behavior of the two networks supports the allegation we proposed in Section 4.3, namely that the HCCA network would be the fastest (in number of epochs) at learning the training set, while the 2CCA network would end up as being the best at generalizing. In the next two sections we shall examine this coherence more thoroughly.

5.4 Experiment 1: Scaling Behavior

In this section we analyze the behavior of the two types of networks as a function of the training set size. Recall that the six training sets we apply contain p = 100, 200, 300, 500, 750, and 1000 training examples, respectively, and that each experiment is carried out ten times with different random weights.

Table 2 shows the average of the most important results obtained for each of the six training sets, while the exact figures of all the experiments discussed in this section can be found in Appendix B.

Figure 22 shows the average performance of the different networks on the test set S_{test}^{1000} as function of p, i.e. it shows the networks' generalization ability as function of the size of the applied training sets. Moreover, the best and the poorest run for each training set are shown as well. Not surprisingly we see that the two curves increase monotonically with the set size p. We also see that the training set S_{train}^{100} contains too few examples to enable the networks to find a satisfactory generalization. Indeed the improvement by

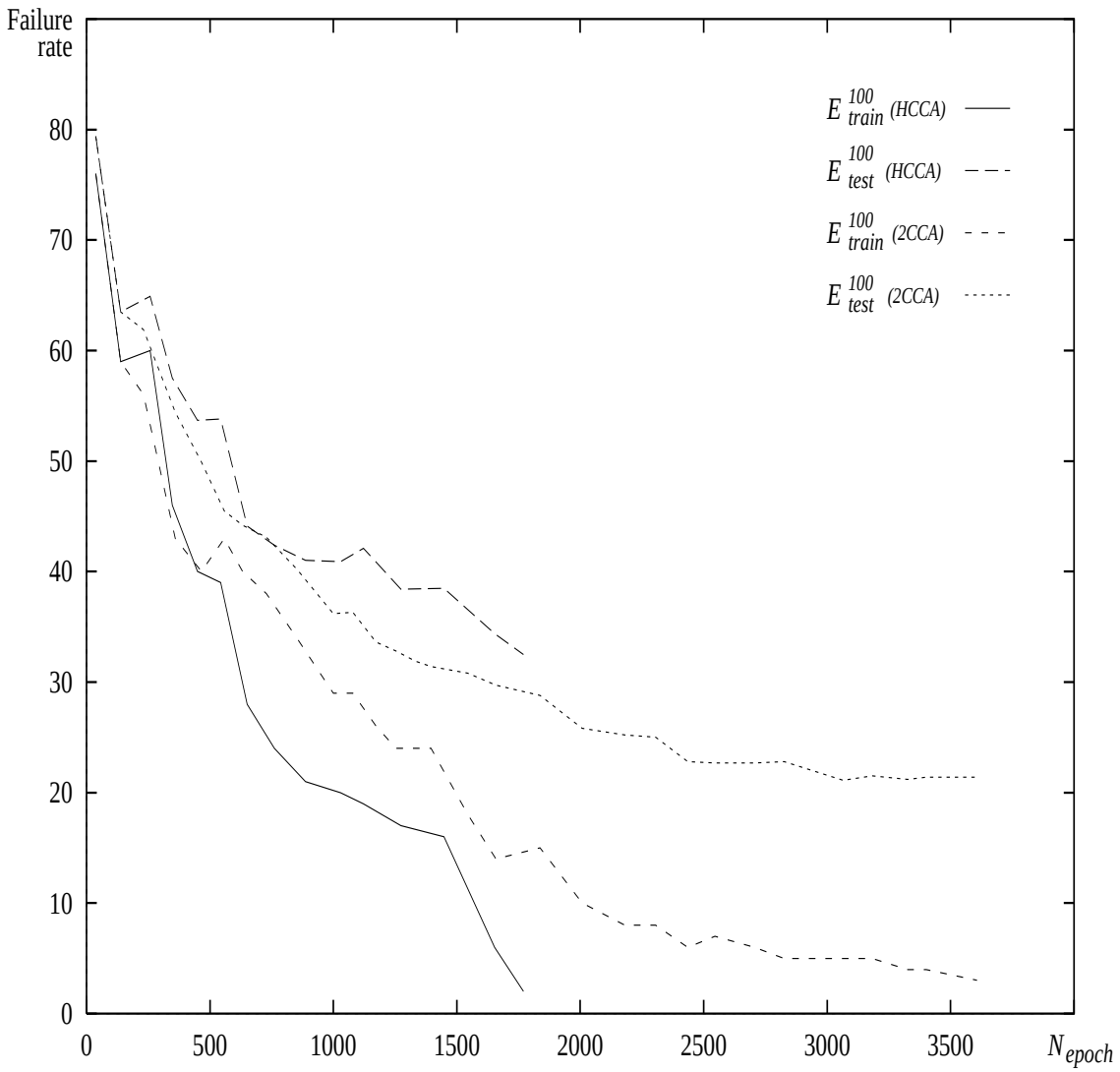

Figure 21: Comparison of evolution and performance of the HCCA and the 2CCA networks. For both networks, the failure rate is depicted as function of the number of epochs. The two lower graphs show the failure rate on the applied training set for the HCCA and the 2CCA network, respectively, while the two upper graphs show the failure rate on the test set.

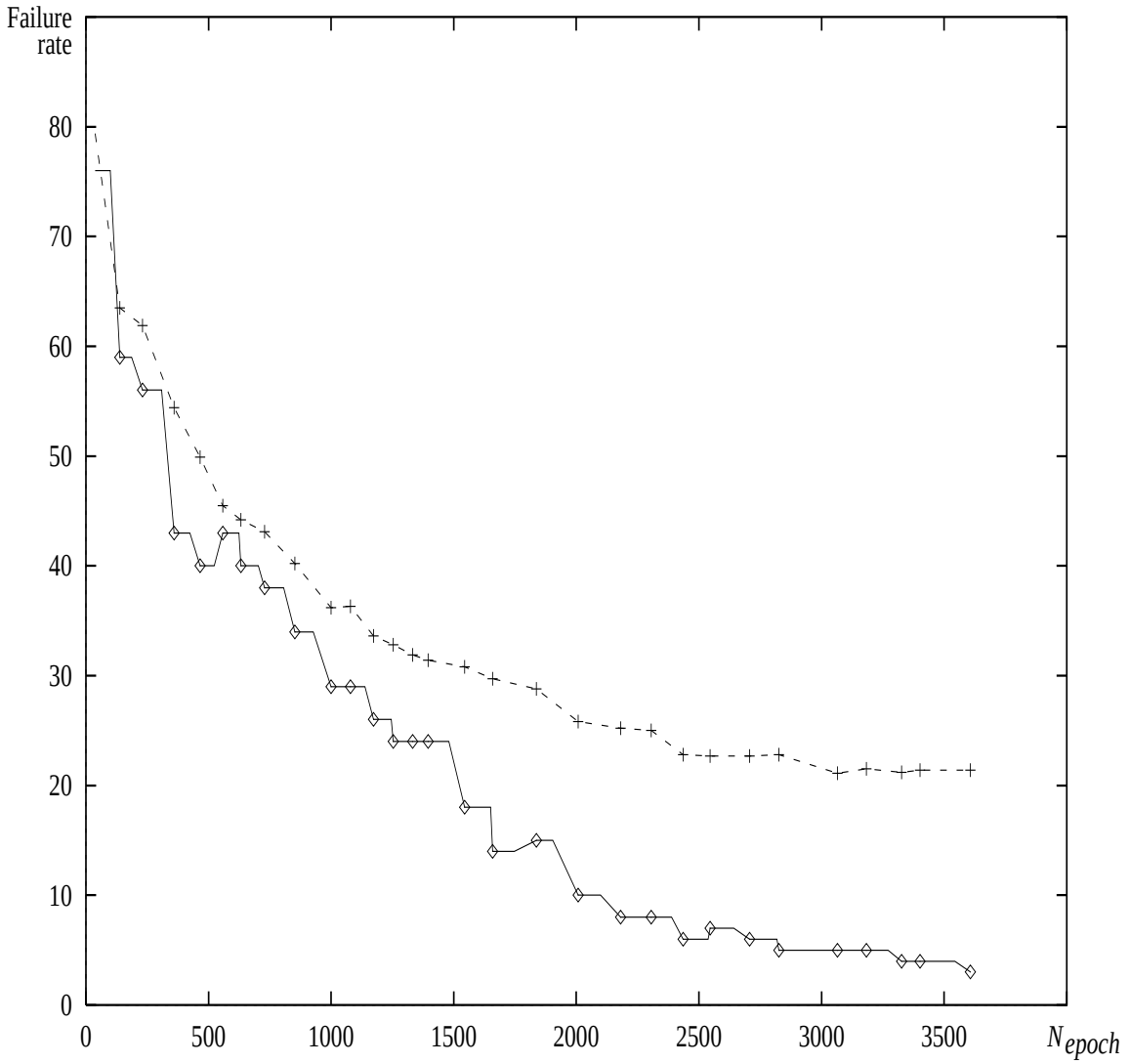

Figure 20: Evolution of a typical 2CCA network that has been trained on S^{100}_{train}. The diamonds ($\diamond$) show the failure rate on the applied training set, i.e. E^{100}_{train} as function of the number epochs, while the plusses ($+$) show E^{100}_{test} as function of the number of epochs.

well if the employed training algorithm builds on the gradient descent technique. The reason is that the function that are minimized usually is defined as an *average* measure of the global error over all patterns and output units. This means that the number of misclassified examples can increase temporarily, even though the global averaged error decreases in each training cycle. In the long run, however, the effect is of course as expected, and the number of misclassified examples will inevitably decrease.

Figure 20 shows that the stopping criteria we have chosen might be a little too strict. The last 4 or 5 hidden units do not achieve much and in this case the network would have performed equally well if training was stopped after approximately 3000 epochs. However, as we mentioned in Section 5.2.2 where we discussed the stopping criteria, there are other cases where the criteria has to be this strict in order to obtain a satisfactory generalization ability. Sometimes the criteria is a little too strict and sometimes it is a little too vague, but on the average the level seems to be appropriate.

Figure 21 summarizes the characteristics of the two networks and enables us to make a temporary comparison of their performance. The two graphs that end at approximately 1700 epochs show the failure rate for the HCCA network, while the failure rate of the 2CCA network is represented by the two remaining graphs.

First of all, we observe that the HCCA network needed only half as many training epochs than the HCCA network to learn the training set. Furthermore, the number of hidden units installed in the HCCA network is only half of the number of units installed in the 2CCA network. This indicates (precisely as expected) that the computational capabilities of the HCCA network are far more powerful than of the 2CCA network.

However, when we focus on the networks' generalization ability, the situation is the quite opposite: The HCCA network is at no time better at classifying the patterns from the test set S_{test}^{1000}. And even after 1700 epochs, where the HCCA network has learned the training set and its generalization ability is at its highest, the 2CCA network outperforms it with almost 4% (28.8% vs. 32.5%). And if we compare the two final networks, the difference is even more distinct: The HCCA network misclassifies 35.5% while the 2CCA network (only) misclassifies 21.4%, i.e. a difference of 14.1%.

For the present we may thus conclude that even though the 2CCA network needs more training epochs to learn the training set (and uses more hidden units) than the HCCA network, its way of expanding the version space seems to be more advantageous. This result supports our discussion in the two previous chapters where we argued for the advantages of finding the simplest possible generalization. To see this, let us focus on the size or dimension of the hypothesis space of the two final networks. According to Equation 5 and 6 on page 38, we find that

$$N_{con}^{\mathrm{HCCA}}(14) \;=\; 3(2+14) \;+\; 14\cdot 2 \;+\; \sum_{j=1}^{13} j \;=\; 167$$

$$N_{con}^{\mathrm{2CCA}}(29) \;=\; 3(2+29) \;+\; 29\cdot 2 \;=\; 151$$

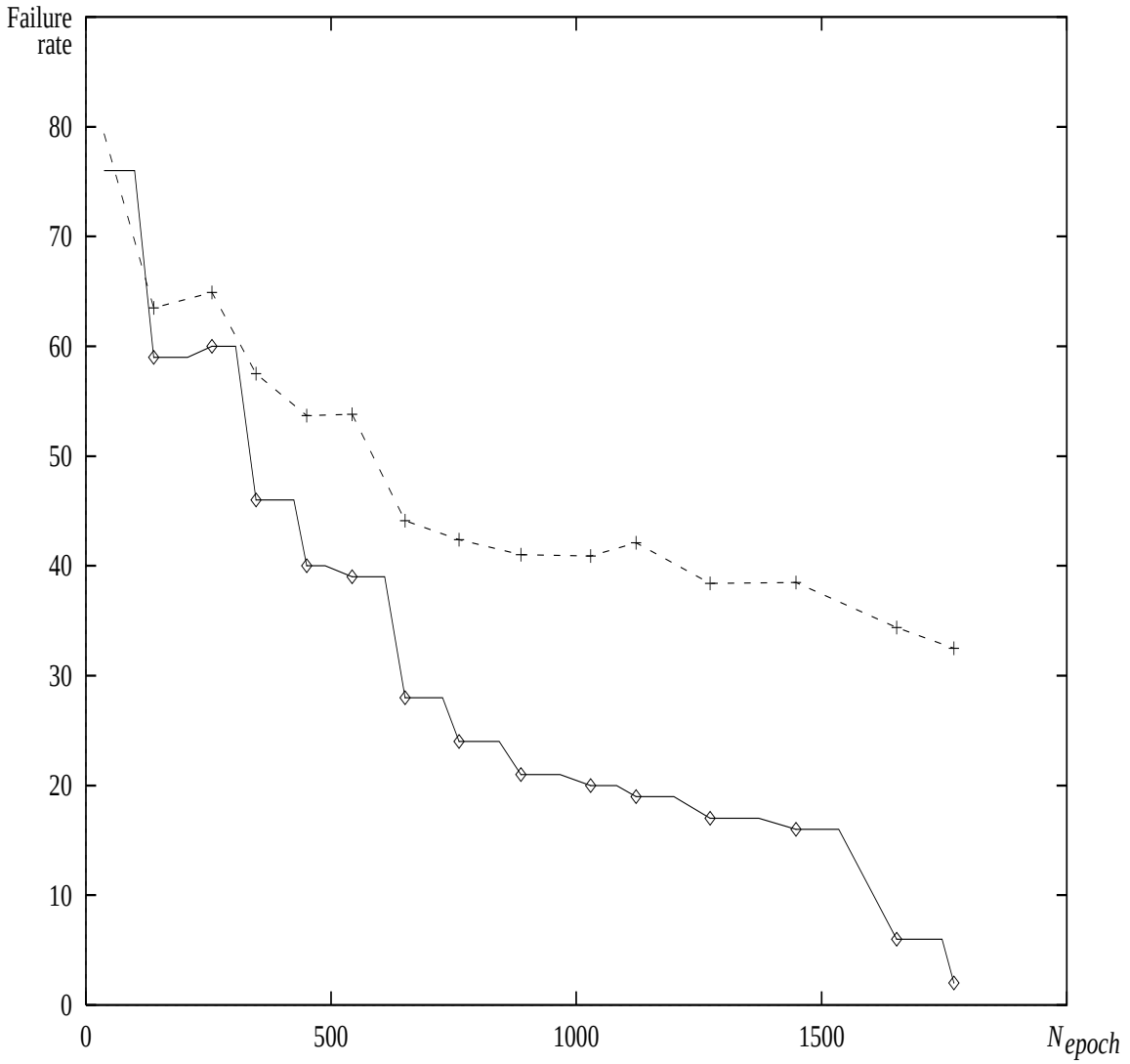

Figure 19: Evolution of a typical HCCA network that has been trained on S^{100}_{train}. The diamonds ($\diamond$) show the failure rate on the applied training set, i.e. E^{100}_{train} as function of the number epochs, while the plusses ($+$) show E^{100}_{test} as function of the number of epochs.

Both networks start with same initial weights (i.e. they start at exactly the same point in weight space) and the performance of the two networks is shown in the Figures 19, 20, and 21.

HCCA: The HCCA algorithm found a solution after 1770 epochs and the generated network has 14 hidden units. This network is able to correctly classify 98% of the training examples, but only 67.5% of the test examples. The exact figures and percentages are given in Table 4 in Appendix A.

Figure 19 shows the failure rate as function of the number of epochs during the simulation. The full-drawn graph shows a detailed picture of the evolution of the network during the training process: All flat or horizontal segments indicate that a new hidden unit is being trained (for as many epochs as shown by the length of the segment) and all non-horizontal segments indicate that the output weights are being trained. That is, the point at the beginning of the flat segments (represented by a $\diamond$) shows the percentage of misclassified training examples (i.e. E^{100}_{train}) by the network (or actually by the sequence of inadequate networks generated) during the training process. As expected the failure rate decreases more or less monotonically with N_{epoch}. The same is true for the dashed graph which shows the failure rate on the test set S^{1000}_{test}, and not surprisingly we see that $E^{100}_{test} > E^{100}_{train}$, especially when N_{epoch} is large.

The effect of the two latest added hidden units should be noticed. The thirteenth unit reduces the percentage of misclassified training patterns from 16 to 6, while the fourteenth hidden unit cuts additional 4% off, resulting in only 2% misclassified training patterns. Furthermore we observe that the failure rate on the test set declines too, although the effect is somewhat more restrained.

This drop in the failure rate after a certain period of smaller and smaller changes seems to appear frequently for these higher-order networks. Besides, this behavior substantiates that the stopping criteria we have chosen is appropriate, since both the learnability and the generalization ability improve considerably when the last few hidden units are added.

2CCA: The 2CCA algorithm used 3607 epochs to find a solution and 29 hidden units were installed. The network correctly classifies 97% of the training examples, and 78.6% of the test examples. The exact figures and percentages are given in Table 5 in Appendix A.

Similar to Figure 19, Figure 20 shows the failure rate on the training set (marked with diamonds) and the test set (marked with plusses) as function of N_{epoch} during the simulation. Again we see that the failure rate on the training set decreases more or less monotonically, although there are some irregularities. Especially the addition of the fifth hidden unit has a negative effect on the classification rate of the training examples. (Again the i'th diamond represents the performance of the network immediately after the i'th hidden unit has been installed.) Notice, however, that the network's generalization ability, in spite of this, is improved with almost 5% when the apparently destructive unit is installed. This temporary negative effect on the classification of training examples is not unusual, and in fact it often appears in static neural networks as

Terminology

We shall apply six different training sets and a single test set. All examples are generated randomly using a uniform distribution over the event space $E = [0,1] \times [0,1]$. We shall use the notation S^p_{train}, $p = 100, 200, 300, 500, 750, 1000$ to denote the six training sets, where p is the number of examples in the sets. Similarly S^{1000}_{test} denotes the test set which contains 1000 test examples. This large number has been chosen to minimize the domain bias in the test set.

5.3 A Typical Run

To illustrate the characteristics of the two dynamic networks and to demonstrate the process of evolution they undergo, we first present a typical run using the smallest training set S^{100}_{train}. The purpose is not to make a direct comparison of the two algorithms (that is the topic of the two following sections), but only to introduce the terminology we are going to use in the "real" experiments, and to show the behavior and evolution of the two networks during an entire training process. Thus we hope that this section will provide the reader a greater comprehension and familiarity with the two algorithms.

Notation

The following notation will be used in the rest of this chapter:

S^{1000}_{test} denotes the test set which consists of 1000 randomly generated test patterns.

S^p_{train} denotes the six training sets, where $p = 100, 200, 300, 500, 750, 1000$ refers to the number of training patterns.

N_{epoch} denotes the number of epochs through the training set.

N_{hid} denotes the number of hidden units installed in the final network.

N_{con} denotes the number of connections in the final network, i.e. the dimension of the hypothesis space.

E^p_{train} denotes the percentage of misclassified patterns (according to the binary method) from the training set S^p_{train}:

$$E^p_{train} = \frac{\#misclass.\ patterns}{p} \cdot 100$$

E^p_{test} denotes the percentage of misclassified test patterns (according to the binary method) when the network has been trained on S^p_{train}:

$$E^p_{test} = \frac{\#misclass.\ test\ patterns}{1000} \cdot 100$$

The Euclidean Method: The output vector $\mathbf{z}^p$ is compared to all eight possible binary valued output vectors in the set $V = \{\, \mathbf{v} = (v_1, v_2, v_3) \mid v_i = \{0,1\}, i = 1,2,3 \,\}$, i.e. the vertices of the three-dimensional hypercube H^3 which represent the eight possible categories defined on page 43. Let $\mathbf{v}' \in V$ be the vector which is closest to $\mathbf{z}^p$ in the usual Euclidean topology of $\mathbb{R}^3$. Then $\mathbf{x}^p$ has been correctly classified if and only if $\mathbf{v}' = \mathbf{y}^p$.

The Binary Method: If $z_i^p \in [0, 0.4] \cup [0.6, 1]$ for $i = 1,2,3$, the output vector $\mathbf{z}^p$ is transformed into a binary valued vector $\mathbf{v}' \in V$ according to the following rule:

$$v'_i = \begin{cases} 0 & , \quad 0 \leq z_i^p \leq 0.4 \\ 1 & , \quad 0.6 \leq z_i^p \leq 1 \end{cases}$$

If and only if $\mathbf{v}' = \mathbf{y}^p$ we define $\mathbf{x}^p$ as correctly classified. This measure focuses on resolution from each of the three output units, and if just one of them is in doubt or hesitates (i.e. $0.4 < z_i^p < 0.6$, for any $i = 1,2,3$), the test pattern is considered as misclassified.

Not surprisingly it turns out that the Euclidean method is the weakest of the two measures, and in general the percentage of correctly classified patterns using this method on a trained network is 2% – 4% higher than when the binary method is used[22]. In the presentation of the obtained results in the appendices, both methods of measuring the performances are shown. However, all the graphs and comparisons presented in the following sections build upon the figures resulting from the binary method solely, as we find this method most realistic and useful.

5.2.4 Training and Test Sets

The composition of the applied training and test sets is yet another potential source that might affect the obtained results. In Section 3.2 on page 24 we discussed the importance of being aware of distinct types of biases, and we mentioned that domain biases, such as a skewed probability distribution of the applied training examples, might affect the search for a good generalization. Especially if the number of training examples is small, this effect is not unlikely. Thus, domain biases is also a weighty reason that one should be cautious about comparing results obtained by different researchers, even though they have applied the same learning algorithm. However, all the comparisons we are going to make between the performances of the two distinct types of networks, will always imply that the networks have been trained on the very same set of examples. This should reduce the domain bias effect considerably.

[22]Before training of the network begins, the difference can be as high as 25%. However, as learning proceeds the two measures approaches each other, and when the training set is learned the difference is reduced to approximately 3%.

the winning candidate unit will be installed and training of all the output weights will commence.

The criterion we have chosen that determines when to stop the training of the network (i.e. when the training set has been learned sufficiently well and a satisfactory generalization is found), is a somewhat weak criteria. Formally it is defined as:

$$\text{ErrorIndex} = \sqrt{\sum_{i=1}^{N_{out}} \sum_{p=1}^{N_{pat}} (z_i^p - y_i^p)^2 / N_{out} N_{pat}} / \text{StdDev}$$

where z_i^p (y_i^p) is the actual (desired) output of unit i when pattern p is presented, respectively, and StdDev is a constant attached to the training set that normalizes the expected outputs. We can think of the ErrorIndex as a measure for the averaged normalized error at a given time.

Some of the exploratory runs we performed were conducted with special reference to finding a proper lower limit for this index. If the limit is too high, the training set will not be learned properly, and and if it is too low the network is forced to learn the training patterns extremely well, and hence an unnecessary large number of hidden units will be installed. And this imply that the weight space –and thereby the version space– will be oversized, and then we know that the probability of overfitting and misclassification of future events will increase. The results from the exploratory experiments showed that a limit in the range from 0.15 to 0.25 would be an appropriate tradeoff, so we decided to use 0.20 as value for the 'errorindexthreshold' parameter. That is, training of the network will proceed as long as the ErrorIndex is greater than 0.20.

Finally, it should be noticed that we have used the asymmetric sigmoid function as output function of both hidden units and output units. The function was implemented as follows:

$$asymsigm(x) = \begin{cases} 0 & , \quad x < -15 \\ 1 & , \quad x > 15 \\ 1/(1 + exp(-x)) & , \quad \text{otherwise} \end{cases}$$

5.2.3 Criteria of Correct Classification

To measure a network's performance on a test set, two distinct methods are applied for determining into which of the eight possible categories a given test pattern has been classified. The first method uses the Euclidean distance between the actual and the eight possible output vectors to find the closest or most similar one, while the other method transforms the actual output vector into a binary vector (if possible) which then is compared with the desired output vector to see if they are identical. Let us define these two methods more formally:

Assume that the test pattern $\mathbf{x}^p$ has been presented to a network, and that the desired binary valued output vector is $\mathbf{y}^p = (y_1, y_2, y_3) \in \{0, 1\}^3$. Finally, let $\mathbf{z}^p = (z_1, z_2, z_3) \in [0, 1]^3$ be the real valued output vector from the network.

Parameter Settings		
Parameter:	**Value:**	**Description:**
errorindexthreshold	0.200	stop training when ErrorIndex < EIT
inputchangethreshold	0.030	fraction that the error must change to count as a significant change
inputdecay	0.000	used by quickprop. Keep weights from growing too big
inputepsilon	1.00	used by quickprop to control amount of gradient descent
inputmu	2.00	used by quickprop to train input weights
inputpatience	8	if we go for this many epochs with no real improvement, stop training of candidate units
ncandidates	8	size of candidate pool
ninputs	2	number of input units
nonrandomseed	FALSE	FALSE => random initialization of weights
noutputs	3	number of output units
outputchangethreshold	0.010	fraction that the error must change to count as a significant change
outputdecay	0.0001	used by quickprop. Keep weights from growing too big
outputepsilon	0.35	used by quickprop to control amount of gradient descent
outputmu	2.00	used by quickprop to train output weights
outputpatience	8	if we go for this many epochs with no real improvement, stop training of output weights
outputtype	ASYMSIGMOID	type of output function of each output unit
sigmoidprimeoffset	0.100	added to sigmoid prime to kill flat spots
unittype	ASYMSIGMOID	type of output function of each hidden unit installed
usecache	TRUE	all errors and output values are cached for the entire training set
weightmultiplier	1.00	scaling factor for candidate correlation
weightrange	1.0	range for random initialization of weights: $[-1, 1]$

Table 1: Parameter settings used for the final experiments.

5.2 Methodology

Basically, all experiments are carried out according to the following three step procedure: First the weights in the initial network are set to some random values in the range from -1 to 1. Next the network is trained on a set of randomly generated training patterns. And finally, the network's performance is tested on a set of random test patterns in order to measure or estimate its generalization ability. Although this procedure is straightforward, there are a number of factors which affect the measure. Especially the various learning parameters, the stopping criteria, the criteria of correct classification, and the large amount of randomness involved in choosing the training patterns and the initial weights could affect the final performance and should be given some consideration.

5.2.1 Initial Weights

First of all, the initial randomly generated weights have a significant effect. We also discussed this issue in Chapter 3, and we found that the learning algorithm starts its search for a good generalization at the point in weight space dictated by the initial weight values. And since we apply a gradient descent algorithm, the only part of the version space that will be explored is the area in the vicinity of this initial point[21]. But since there usually are many minima or hypotheses consistent with the training set, one can see that different initial states may lead to completely different generalizations. One way to minimize this effect is to carry out the same experiment several times and average the obtained results. We have used this method, and unless anything else is mentioned all experiments are repeated ten times.

5.2.2 Stopping Criterion and Learning Parameters

Before we started on the real experiments, we performed a sequence of exploratory runs to find a proper range for a number of parameters. We will not go into depth about an explanation of all the distinct parameters, but only list the parameter settings we applied for the final experiments. See Table 1. For a thorough discussion and explanation of the functioning of the parameters, see [Fahlman *et al.*] and [Fahlman].

Besides these parameter settings, we used an upper bound of 200 as the maximum number of epochs that could be used to train the pool of candidate units, and as the maximum number of epochs that could be used to adjust the output weights in the primary network before a new hidden unit should be installed.

Notice that both the input and the output patience parameter are set to 8, i.e. if there is no significant change of the output error (determined by the two changethreshold parameters) for 8 epochs, either a new pool of 8 candidates will be generated (in case the output weights were being trained), or (if the candidate units were being trained)

[21]This should be taken with a grain of salt. Since the version space is expanded dynamically during the search, this is not quite true. Nevertheless, it is just a smaller part of the version space that actually will be explored.

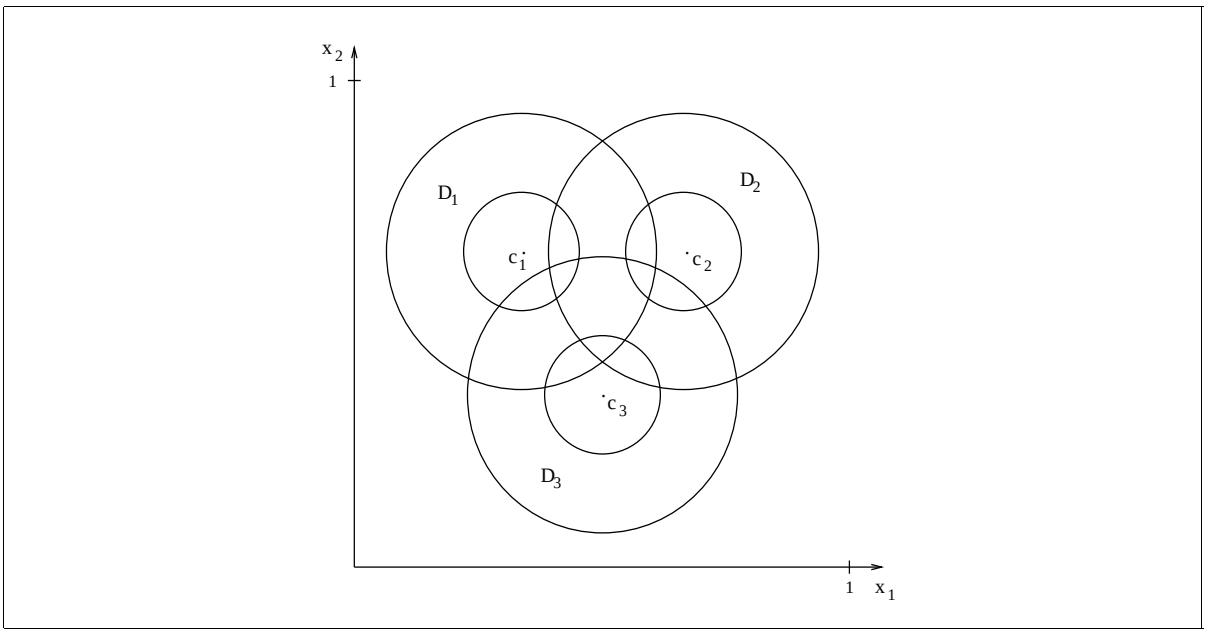

Figure 18: Geometric depiction of the three-disc problem.

Whenever a two-dimensional event vector $\mathbf{x}$ is presented to a network, the task is thus to correctly classify $\mathbf{x}$, i.e. to determine in which of the eight possible categories $\mathbf{x}$ lies:

$$D_1^C \cap D_2^C \cap D_3^C\,, \quad D_1 \cap D_2^C \cap D_3^C\,, \quad D_1^C \cap D_2 \cap D_3^C\,, \quad \cdots \quad, D_1 \cap D_2 \cap D_3$$

where $D_i^C = E \setminus D_i\,,\ i = 1, 2, 3\,.$

One straightforward way to encode this desired input/output behavior, is to apply a *local* or *positional encoding* of the three classes D_1, D_2, and D_3, i.e. to let a single output unit represent each of the three classes. Thus, if $\mathbf{x} \in D_1 \cap D_2 \cap D_3^C$, the desired output vector should be (1,1,0), and if $\mathbf{x} \in D_1^C \cap D_2^C \cap D_3^C$, a (0,0,0) output vector should be produced.

Notice that the problem is somewhat more complicated than it might appear at a first thought. The $D_1^C \cap D_2 \cap D_3$ region e.g., consists of a union of two disjoint sub-regions of which the one is very small. And similarly, the $D_1^C \cap D_2^C \cap D_3^C$ region is a union of four disjoint sub-regions.

In the following experiments all training and test data are generated randomly using a uniform distribution over the event space. Thus, the number of generated instances of a certain category represents a statistical measure of the area of the corresponding region.

We believe that this geometric classification problem meets all the requirements we proposed above. Especially we find the problem well-suited for testing the generalization ability of the trained networks, and indeed training and test data can be generated en masse.

In order not to make some experiments which build upon these drawbacks, we have decided to arrange a series of experiments which primarily should be analyzed and interpreted within the scope of this thesis, and not used for a direct comparison with results obtained by other researchers. Therefore we have chosen not to focus on the two-spirals benchmark problem which Fahlman and Lebiere used as the primary benchmark for their study, when they presented the Cascade-Correlation Learning Algorithm [Fahlman *et al.*]. Besides, Fahlman and Lebiere report that the best two-spirals performance obtained to date using the backpropagation algorithm, is by a 2-5-5-5-1 feed-forward network with shortcut connections between all four layers, i.e. each unit receives an incoming connection from every unit in *every* earlier layer, not only from the immediately preceding layer. This indicates that the problem is quite difficult for a two-layer network to solve, and thus the networks generated by the 2CCA algorithm would not have a fair chance against the HCCA networks, which seem to perform extremely well on this problem[20].

Instead we decided to find a classification problem which possesses the following properties:

- it should be a nontrivial problem which demands at least one layer of hidden units, i.e. the problem should not be linearly separable.
- it should be well-suited for measuring the generalization ability of the considered networks, and finally
- the training and test data should be inexpensive and accessible on a large scale.

These properties exclude most of the well-known benchmark problems, so we decided to construct a geometric classification problem ourselves, which we call the *three-disc problem*: In the two-dimensional plane, the unit square is the continuous input domain, i.e. the event space $E = [0,1] \times [0,1]$. (See Figure 18). In E we have three overlapping discs or toruses which represent three different classes D_1, D_2, and D_3.
For $i = 1,2,3$, $D_i \subseteq E$ is defined by:

$$D_i = \{ \mathbf{x} \in E \mid r_{min} < \|\mathbf{c}_i - \mathbf{x}\| \leq r_{max} \}$$

where

$$\begin{aligned} \mathbf{c}_1 &= (0.33, 0.65) \\ \mathbf{c}_2 &= (0.67, 0.65) \\ \mathbf{c}_3 &= (0.50, 0.35) \end{aligned}$$

and

$$\begin{aligned} r_{min} &= 0.1230 \\ r_{max} &= 0.2869 \end{aligned}$$

[20]Actually we have tried to solve the two-spirals problem using the 2CCA algorithm but, as expected, the result was disappointing: When 100 hidden units had been installed, the network was only able to learn approximately half of the training instances. There is no doubt that this problem is well-suited for networks that are able to compute higher-order correlations, as the HCCA networks e.g.

5 Empirical Analyses

In the first two chapters of this thesis, we examined analytically the capabilities of two-layer networks and, inspired by traditional Machine Learning and the seminal work by Mitchell, we presented a different conceptual approach to the process of generalization in the general category of feed-forward networks. Furthermore, using this framework we outlined some of the nice properties and behavior of dynamic neural networks, and, still analytically, we argued for the advantage of employing the network construction method as an efficient technique for finding a well-generalizing network tailored for the particular problem considered. In the previous chapter we became somewhat more concrete and we carefully presented and discussed two specific dynamic learning algorithms, which during the search for a good generalization employ two distinct network construction techniques to incrementally expand the hypothesis space. In this chapter we shall go the whole length, and examine and compare these two dynamic learning algorithms empirically.

The main purpose is to investigate experimentally how the networks' generalization ability is affected by the two distinct network construction strategies, i.e. how the generalization ability is affected by the two distinct ways of incrementally increasing the complexity of the hypothesis representation in the two network types. By that means we hope to be able to achieve a better insight in the advantages and disadvantages of higher-order feature detectors, and to obtain an improved understanding of the process of generalization in artificial neural networks in general.

In Section 5.1 below, we describe the classification problem we have chosen to consider and which shall make the basis of the subsequent empirical comparison of the two learning algorithms. In Section 5.2 we discuss the methodology underlying the experiments, and in Section 5.3 a typical run of each of the two learning algorithms is presented. The purpose of this is to introduce the terminology that is used in the following experiments, and to familiarize the reader with the two algorithms. Section 5.4 is devoted to an analysis of the scaling behavior of the two network types, and finally in Section 5.5, the importance of the hypothesis representation is analyzed. The topic of interest is to examine how the complexity of the hypotheses that the networks are able to form during the training process affects their learnability and especially their generalization ability.

5.1 Description of the Three-Disc Problem

An important question in the field of artificial neural networks is how to measure one learning algorithm against another. When a new learning algorithm is proposed, the typical approach is to present one or two experiments that compare the new algorithm against, say, the standard backpropagation learning algorithm. Unfortunately, no two researchers seem to run the same problem in the same way, and, not surprisingly, almost everyone outperforms backpropagation by at least one order of magnitude. Of course backpropagation, or any other learning algorithm, is very sensitive to the choice of training parameters, so whenever a new learning algorithm is proposed and a test problem is chosen, there is always the question of whether backpropagation was given a fair chance.

HCCA networks (especially when p is large), and we believe therefore that the HCCA networks will have a poorer generalization ability than the 2CCA networks.

The Significance of the Information

The other reason why the HCCA units potentially are much more powerful than the 2CCA units, is the increasing complexity of the information they are able to retrieve from the previous (or lower-lying) hidden units. By the XOR example on page 27 we demonstrated that from the p'th hidden unit's point of view, we can think of all the previous $p-1$ hidden units as performing projections of some of the presented input patterns onto a space (or hypercube vertex) of a higher dimension. That is to say that the previously installed hidden units perform projections of projections of $\cdots$, and therefore, especially when $p \gg n$, the p'th unit is presented to input patterns of which the main part consists of a transformation or complex mapping of the original input onto a space of a considerable higher dimension. In other words, the main part of the information that is presented to the p'th hidden unit has been preprocessed by all the lower-lying hidden units, and it contains therefore high-order correlations.

If we now recall that the dimension of the hyperplane that each hidden unit defines in weight space is determined by the number of incoming weights to that unit, we see that the dimension of the hyperplanes created by the HCCA networks increases linearly with the number of hidden units installed. The first hidden unit defines a hyperplane separation in the n dimensional event space E itself, which in the continuous case might be $\mathbb{R}^n$, i.e. $E = \mathbb{R}^n$. The second hidden unit defines a hyperplane separation in a $n+1$ dimensional space E_2, which is defined by the cross product of the original event space and the one-dimensional classification space (or projection space) determined by the decision region of the first hidden unit, i.e. $E_2 = \mathbb{R}^n \times f_1(\mathbb{R}^n)$. The third hidden unit defines a hyperplane separation in a $n+2$ dimensional space E_3, defined by the cross product of the original event space, the classification space of the first hidden unit, and the classification space of the second hidden unit, i.e. $E_3 = \mathbb{R}^n \times f_1(\mathbb{R}^n) \times f_2(\mathbb{R}^n, f_1(\mathbb{R}^n))$. Continuing this expansion scheme, we see that both the *volume* and the *significance* of the information gradually increase. There is therefore reason to believe that the following experiments will show that the HCCA networks are able to learn the applied training sets faster than the 2CCA networks, when they are faced with the same problem. (By "faster" we mean that fewer hidden unit need to be installed, and that the necessary number of training epochs will be considerable smaller.)

However, when we focus on their ability to generalize, we believe the situation will turn out to be the opposite, i.e. the 2CCA networks will outperform the HCCA networks. And the reason for this can be found in our discussion of generalization in Chapter 3 where we argued for the advantage of finding the simplest possible generalization: Among other things we focused on the effects of employing a simple versus a complex hypothesis representation (such as using one or two rectangles), and how the number of degrees of freedom of a network affects its ability to generalize. And in both cases we found that restrictions and limitations were preferable: If unnecessary complex hypothesis representations were allowed, the version space would be "oversized," and hence the possibility of misclassification would increase. And similarly, if the number of weights or free parameters in a network was larger than the number of constraints imposed by the training instances, the possibility of overfitting would increase. And from the discussion above, we know that precisely these two properties are connected with the dynamic construction of

Now, let us use this framework to illustrate how the two network construction algorithms incrementally expand the hypothesis space during the search for a good generalization. The functionality of the two distinct types of hidden units shows clearly that the hypothesis space of a HCCA network expands much faster than the hypothesis space of a 2CCA network, and that is the major reason why the HCCA networks are able to deal with much more complex hypotheses than the 2CCA networks. There is, however, a deeper issue here which is very important for the understanding of the two different ways of forming powerful hypothesis representations. The reason is not only that the weight space expands faster in a HCCA network, but also *how* it expands. Let us for a moment consider each of these causes separately.

The Volume of the Information

Suppose that either networks are faced with the same problem, and that both of them at a certain time have installed p hidden units. Then the number of connections in the two networks (i.e. the dimension of their hypothesis spaces) are respectively:

$$\begin{aligned} N_{\text{con}}^{\text{HCCA}}(p) &= \sum_{i=1}^{N_{out}}(n+p) + \sum_{j=1}^{p}(n+j-1) \\ &= N_{out}(n+p) + p\cdot n + \sum_{j=1}^{p-1} j \qquad (5) \\ N_{\text{con}}^{\text{2CCA}}(p) &= \sum_{i=1}^{N_{out}}(n+p) + \sum_{j=1}^{p}(n) \\ &= N_{out}(n+p) + p\cdot n \qquad (6) \end{aligned}$$

where n is the number of input terminals, i.e. the dimension of the event space.

We notice that when the p'th unit is being installed, the hypothesis space of the HCCA network is expanded with $N_{out}+n+p-1$ extra dimensions, while the hypothesis space of the 2CCA network is expanded with only $N_{out}+n$ extra dimensions, i.e. a difference of $p-1$. In other words, the dimension of the hypothesis space of a HCCA network grows quadratically, while the dimension of the hypothesis space of a 2CCA network only grows linearly. So, especially when p is large, the expansion of the HCCA hypothesis space is considerable. Furthermore we notice that it is the number of incoming connections that is interesting: When the p'th hidden unit is being installed in a HCCA network, it receives $n+p-1$ incoming connections, while it only receives n connections in a 2CCA network. Therefore the volume of information that the p'th hidden unit receives grows linearly with p in a HCCA network, while it is constant (or independent of p) in a 2CCA network.

$$\tilde{f}_j^{\mathrm{HCCA}} = \begin{cases} f_1^{\mathrm{HCCA}}(\mathbf{x}) & , \quad j = 1 \\ f_j^{\mathrm{HCCA}}(\mathbf{x}, \tilde{f}_1^{\mathrm{HCCA}}, \tilde{f}_2^{\mathrm{HCCA}}, \cdots, \tilde{f}_{j-1}^{\mathrm{HCCA}}) & , \quad 1 < j \leq N_{hid} \end{cases}$$

To illustrate how the complexity of these functions gradually increase, let us take a closer look at the expansion scheme (to keep things simple, we have omitted the HCCA index):

$$\begin{aligned} \tilde{f}_1 &= f_1(\mathbf{x}) \\ \tilde{f}_2 &= f_2(\mathbf{x}, \tilde{f}_1) = f_2(\mathbf{x}, f_1(\mathbf{x})) \\ \tilde{f}_3 &= f_3(\mathbf{x}, \tilde{f}_1, \tilde{f}_2) = f_3(\mathbf{x}, f_1(\mathbf{x}), f_2(\mathbf{x}, f_1(\mathbf{x}))) \\ &\vdots \\ \tilde{f}_{N_{hid}} &= f_{N_{hid}}(\mathbf{x}, \tilde{f}_1, \tilde{f}_2, \cdots, \tilde{f}_{N_{hid}-1}) = \cdots \end{aligned}$$

It is now easy to see how the complexity of these functions gradually increase due to the nested recursivity and the increasing number of arguments. Indeed the latest added units have a potentiality to evolve into very powerful high-order feature detectors.

The function $\tilde{h}_i^{\mathrm{HCCA}}$ computed by the i'th output unit can be expressed similarly:

$$\tilde{h}_i^{\mathrm{HCCA}} = h_i^{\mathrm{HCCA}}(\mathbf{x}, \tilde{f}_1, \tilde{f}_2, \cdots, \tilde{f}_{N_{hid}}) \quad , \quad 1 \leq i \leq N_{out}$$

2CCA: Let $\mathbf{x}$, N_{hid}, and N_{out} be defined as above. Then the function computed by the j'th hidden unit, $\tilde{f}_j^{\mathrm{2CCA}}$, and the function computed by the i'th output unit, $\tilde{h}_i^{\mathrm{2CCA}}$, is determined by:

$$\tilde{f}_j^{\mathrm{2CCA}} = f_j^{\mathrm{2CCA}}(\mathbf{x}) \quad , \quad 1 \leq j \leq N_{hid}$$

$$\tilde{h}_i^{\mathrm{2CCA}} = h_i^{\mathrm{2CCA}}(\mathbf{x}, \tilde{f}_1, \tilde{f}_2, \cdots, \tilde{f}_{N_{hid}}) \quad , \quad 1 \leq i \leq N_{out}$$

Clearly, these functions are much less complex, and the computational power of the hidden and the output units is restricted to first- and second-order correlations, respectively.

We can also illustrate and compare the complexity of the two distinct types of hidden units in another way, namely by focusing on the functionality. To keep things simple, let us for a moment assume that the output function of the N_{hid} hidden units in either networks is the threshold function defined in Section 2.2 (i.e. $out_j(a_j) = sgn(a_j), 1 \leq j \leq N_{hid}$). Then we have:

$$f_j^{\mathrm{HCCA}} : \mathbb{R}^n \times \underbrace{\{0,1\}^1 \times \cdots \times \{0,1\}^1}_{j-1} \mapsto \{0,1\}^1$$

$$f_j^{\mathrm{2CCA}} : \mathbb{R}^n \mapsto \{0,1\}^1$$

step-size problem by using a second-order technique which builds on the parabola idea described above.

- The moving target problem is another source of inefficiency in BP learning in static networks. "Briefly stated, the problem is that each unit in the interior on the network is trying to evolve into a feature detector that will play some useful role in the network's overall computation, but its task is greatly complicated by the fact that all the other units are changing at the same time. [*omitted text*] The error signal defines the problem that the unit is trying to solve, but this problem changes constantly. Instead of a situation in which each unit moves quickly and directly to assume some useful role, we see a complex dance among all the units that takes a long time to settle down." [Fahlman *et al.*, p. 2].

  The HCCA and the 2CCA algorithms use an extreme technique to combat this problem, namely by allowing only one hidden unit to evolve into a feature detector at a time. This eliminates totally the moving target effect (or the cascade dance), and any unit that is not frozen can quickly choose some useful role in the overall solution and then move decisively to fill that role.

Before we proceed to a comparison of the two cascor algorithms, we want to emphasize two additional reasons for their ability to learn fast from examples. Firstly, while the output weights are being trained, all the other weights in the network are frozen, and similarly, while the candidate weights are being trained, all weights in the primary network are temporarily frozen. That is, at any given time, only one layer of weights is trained while the rest of the network is fixed, and it is therefore possible to record all unit-values and output errors repeatedly, and then reuse these cached values during the training instead of recomputing them each time.

Secondly, the single layer of adjustable weights also implies that the quickprop algorithm converges very fast, as there is no need to propagate error signals backwards through the network. Especially when the candidate units are being trained this is important because the residual error signal can be broadcasted directly to all the candidates in each iteration.

4.3.1 Analysis of the Two Distinct Hypothesis Space Construction Techniques

As mentioned above, the HCCA algorithm has its strength in its ability to construct very powerful (and complex) high-order feature detectors, while the 2CCA algorithm is more straightforward in the way that all added units are first-order feature detectors. Let us use the notation from Chapter 2 and the XOR example on page 27 to formalize this for each of the two algorithms:

HCCA: Suppose that at a given time in a particular training session, N_{hid} hidden units have been installed in a network which has N_{out} output units. Let $\mathbf{x} \in \mathbb{R}^n$ denote a n-dimensional input vector in the event space. Then the function computed by the j'th hidden unit, $\tilde{f}_j^{\text{HCCA}}$, is determined by the following recursive expression:

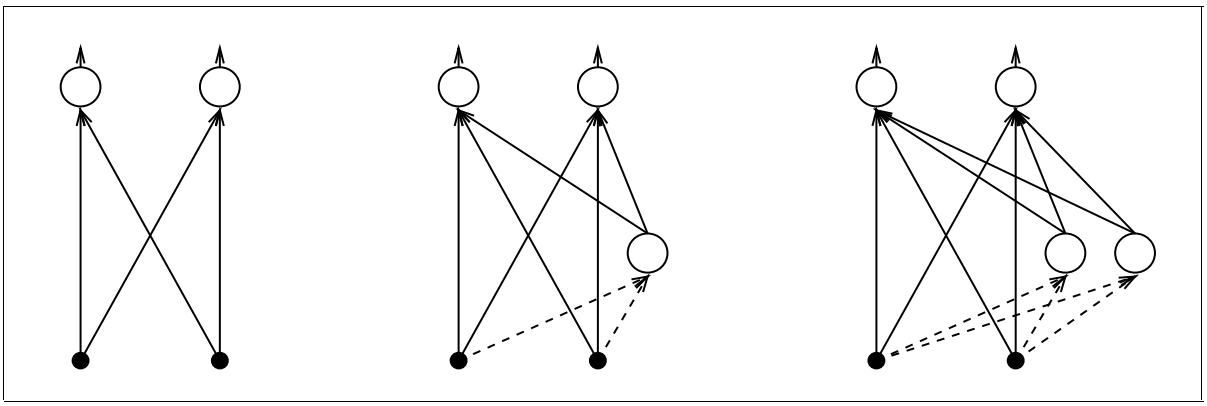

Figure 17: Evolution of the second-order cascor architecture. All hidden units are installed at the same level, namely as first-order feature detectors. Dashed lines represent permanently frozen weights while the full-drawn lines are trained repeatedly.

4.2.2 The Second-Order Cascor Algorithm

The second-order cascor algorithm (2CCA) is identical to the HCCA algorithm, except for the network construction strategy that it employs. The only difference is that all the candidate units that are trained and installed using the 2CCA algorithm, are connected to the n original input terminals only. This is illustrated in Figure 17. It is now clear that all the hidden units that are trained and installed by the unit-creation algorithm are fed with exactly the same information from the event space, and as their incoming weights are frozen when they are added to the network, they will therefore all end up as permanently specialized first-order feature detectors.

4.3 The Two Cascade-Correlation Learning Algorithms Revisited

First, let us focus on some of the major advantages of the two algorithms described above. The HCCA algorithm was developed in an attempt to overcome certain problems and limitations of the popular backpropagation (BP) learning algorithm, especially the slow pace at which it learns from examples. Fahlman and Lebiere identified two such major problems and called them the *step-size problem* and the *moving target problem.*

- The step-size problem occurs because the BP algorithm uses only the partial first-order derivative of the overall error function with respect to each weight, to perform a gradient descent in weight space. This technique imply that we take infinitesimal steps down the gradient vector in order to reach a local minimum of the error function.

  For fast learning, however, we want to take the largest possible step in each iteration. Unfortunately, the largest possible step size depends on the curvature in the vicinity of the current point in weight space, i.e. some information about higher-order derivatives is needed at each point, and this information is not available in the standard BP algorithm. The quickprop algorithm, on the other hand, overcomes the

o_c^p	is the candidate unit's output on pattern p
$\overline{o}_c$	is o_c^p averaged over all patterns
E_i^p	is the residual output error on pattern p observed at output unit i, and
$\overline{E}_i$	is E_i^p averaged over all patterns

That is, the goal is to maximize the sum over all output units i of the magnitude of the "correlation" between the candidate unit's output o_c and the residual error E_i observed at output unit i.

In order to maximize $corr_c$, $\partial corr_c/\partial w_{c,j}$, the partial derivative of $corr_c$ with respect to each of the incoming weights $w_{c,j}$, must be computed. We find that

$$\partial corr_c/\partial w_{c,j} = \sum_{i,p} \sigma_{c,i}^p (E_i^p - \overline{E}_i) out'^p_c a_{c,j}^p \tag{4}$$

where

$\sigma_{c,i}^p$	is the sign of the correlation between the candidate unit's output value and the i'th output unit's value
out'^p_c	is the derivative of the candidate unit's output function with respect to the sum of its inputs, and
$a_{c,j}^p$	is the input the candidate unit receives from unit j

After $\partial corr_c/\partial w_{c,j}$ has been computed for all incoming weights j, we can perform a gradient ascent to maximize $corr_c$ by using the quickprop algorithm described above. And once again we are training only a single layer of weights. However, Equation 4 may have several local maxima and therefore the candidate unit can get stuck. So, in order to improve the chance of finding a near-optimal feature detector, a number of candidate units should be trained simultaneously each time the network has to be expanded. The use of a pool of candidates is thus beneficial in the way that many parts of the weight space (i.e. hypothesis space) can be explored simultaneously, and therefore the chance of finding the smallest possible network (and simplest possible generalization) is improved. Fahlman reports that a number of empirical experiments demonstrate this phenomenon. If the candidate pool is increased from 8 to 16 say, several epochs is knocked off the average needed for a solution to a given problem. On a serial machine, however, this takes approximately twice as long time to run, so one has to decide what sort of trade-off is wanted for the particular problem considered.

The key idea in focusing on the correlation measure whenever the candidate units are trained, is inspired by statistical correlation analysis: If two signals have a correlation of 1, they differ only in a scale factor. It is therefore clear that if we could develop a new unit whose output correlates perfectly with the residual error, we could plug that unit into the network and, by adjusting the output weights appropriately, cancel all the remaining error. Through experiments Fahlman and Lebiere discovered, however, that in real situations, where only a partial match is possible, the covariance measure works better in networks with sigmoid output units. The reason is possibly that as long as the sign of the correlation is right, the magnitude of the output is less important, since the output connections will ensure that the output units are going to saturate anyway.

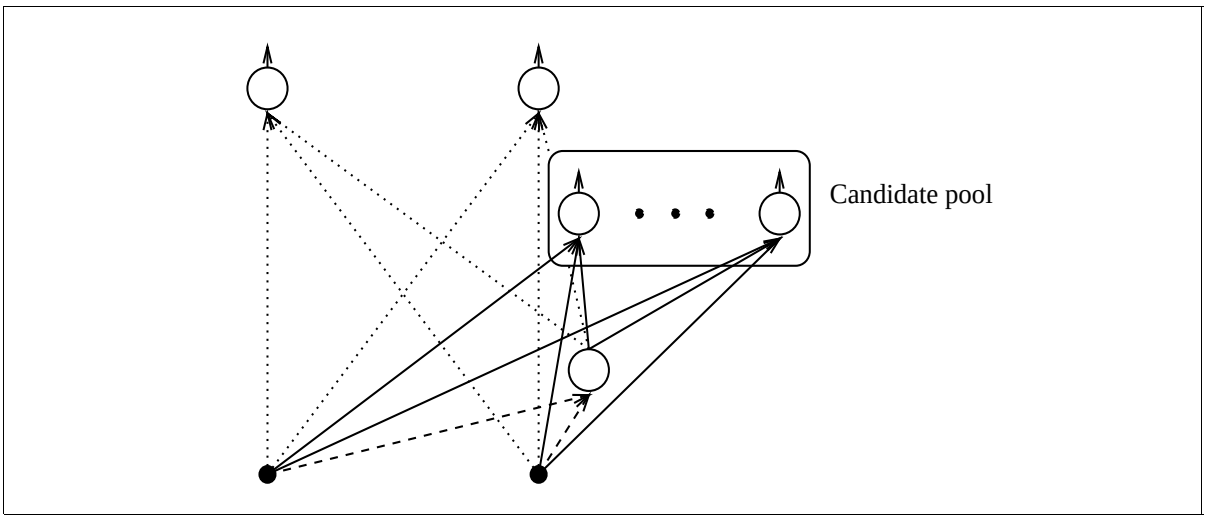

Figure 16: Training of candidate units. The unit-creation algorithm is training a pool of independent candidate units simultaneously (full-drawn lines) while all the output connections in the primary network are temporarily frozen (dotted lines). The dashed lines represent the permanently frozen incoming connections to all pre-existing hidden units. The candidate unit whose output exhibits the maximum correlation with the residual error in the primary network is added by freezing its incoming weights (making it a permanent feature detector) and by establishing a connection to all output units in the primary network.

method, to update the weights. The basic idea builds on the assumption that the error function with respect to each weight is locally quadratic: For the update of weight $w_{i,j}$ at time t, quickprop keeps a copy of the previous error slope, $S_{i,j}(t-1)$, the current error slope, $S_{i,j}(t)$, and the previous weight change, $\Delta w_{i,j}(t-1)$. The two slopes and the step between them are then used to define a parabola, and the current weight update is determined as the minimum point of this curve. Formally, the updating rule of weight $w_{i,j}$ at time t is

$$\Delta w_{i,j}(t) = \Delta w_{i,j}(t-1)\frac{S_{i,j}(t)}{S_{i,j}(t-1) - S_{i,j}(t)}$$

Due to the approximation mentioned above (and a few others), the update algorithm is somewhat more complicated than presented here. Especially if it turns out that $S_{i,j}(t) < S_{i,j}(t-1) < 0$ or $0 < S_{i,j}(t) < S_{i,j}(t-1)$ the formula above has to be modified. For further details, see [Fahlman].

In networks with no hidden units, it can be proven that the sum-squared-error measure has only a single minimum, and therefore the Widrow-Hoff algorithm (or the delta rule) cannot get stuck. The correlation measure used in this algorithm can, however, have several local maxima: Whenever a new hidden unit has to be installed, the adjustment of the c'th candidate unit's input weights is determined from the maximization of the expression

$$corr_c = \sum_{i=1}^{N_{out}} \left| \sum_{p=1}^{N_{pat}} (o_c^p - \overline{o}_c)(E_i^p - \overline{E}_i) \right| \tag{3}$$

where

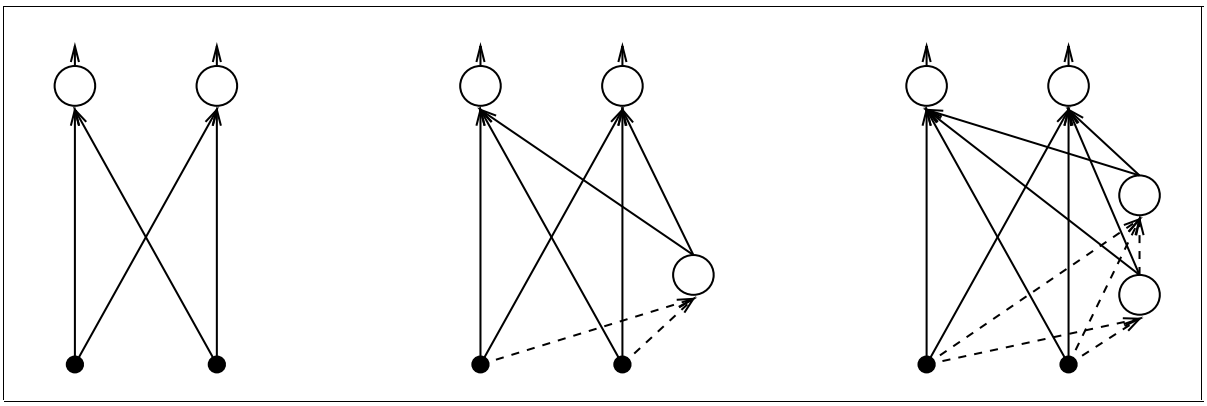

Figure 15: Evolution of the high-order cascor architecture. To the left the initial state and to the right two hidden units have been added. Dashed lines represent permanently frozen weights while full-drawn lines are trained repeatedly.

4.2.1 The High-Order Cascor Algorithm

The high-order cascor algorithm (HCCA) finds an adequate network architecture as follows: Initially it starts with a one-layer fully-connected network consisting of n input and m output units only. See Figure 15. This initial network is a single-layer Perceptron with m output units, where n and m are dictated by the chosen encoding of the environment for the problem considered. This Perceptron is trained as well as possible over the entire training set, using a fast variation of the backpropagation algorithm called quickprop (see below). If no significant error reduction has occurred after a certain number of training epochs (determined by a "patience" parameter set by the user) and we are not yet satisfied by the network's performance, all the weights in the existing network are frozen and a unit-creation algorithm takes over: A pool of candidates is generated. Each candidate unit receives a connection (randomly initialized) from all the input terminals and from all pre-existing hidden units in the temporarily frozen network. The output of the candidate units are not yet connected to the primary network, i.e. the candidate units constitute a number of independent single-layer Perceptrons each having one output unit. See Figure 16. All the candidate units are therefore trained in parallel, again using the quickprop algorithm. The goal is to maximize the magnitude of the correlation[19] between the candidate units' output value, and the residual output error produced by the primary network. The unit from the candidate pool whose output exhibits the maximum correlation is added to the primary network, and the rest of the candidate units are discarded. The selected candidate unit's input weights are frozen and a connection to all the output units in the primary network is established. Thus the unit has now become a permanent feature detector in the network, available for producing outputs or for creating other, more complex feature detectors. When the new unit is installed, all the output weights are once again trained using quickprop. This cycle repeats until the error is acceptable or we give up.

The quickprop algorithm [Fahlman] uses a second-order method, related to Newton's

[19]Actually, it is the covariance, rather than the correlation that is maximized. Fahlman and Lebiere discovered through experiments that the covariance measure actually worked better than the correlation measure, which theoretically ought to be preferable. We shall return to this issue a little later.

NETtalk is a two-layer network trained to convert English words to the corresponding phonemes, and it turned out that the generalization found by NETtalk allowed it to cluster phonemes into a hierarchy according to their similarities [Rosenberg]. Actually, the phonemes were clustered depending on whether they were vowels or consonants, according to the place of articulation, etc. Thus, the features discovered by NETtalk for classifying phonemes were similar to the features that people normally use.

The main purpose of the experiments we are going to carry out in the next chapter, is to examine this quality of high-order feature detectors in more detail, by investigating empirically whether multi-layer network structures really are preferable to the more traditional structure consisting of only two layers, i.e. whether the use of high-order feature detectors really is a good idea or not. Note that this in many ways is a nontrivial question. For example, we must be fully aware of whether we focus on the networks' ability to generalize, or only on their ability to learn the training set. But also the number of epochs needed to train the networks, and the composition of the training set should be taken into account.

4.2 Description of Two Cascade-Correlation Learning Algorithms

Basically, constructive learning algorithms offer a means to adaptively determine both the necessary network connectivity and the weights to solve specific tasks through supervised learning. However, such algorithms are more complex than "ordinary" algorithms that train static neural networks (e.g. the Perceptron Algorithm [Rosenblatt], and the Backpropagation Algorithm [PDP]), since they have to deal with mechanisms for making a variety of extra decisions as for example when it pays to add a new unit to the network instead of continuing to adjust the weights in the existing network and, eventually, where and how to add a new unit.

As mentioned above, the two algorithms we shall experiment with in this chapter use exactly the same learning procedure to train the networks they construct during the learning process. The network construction schemes they employ, however, are quite diverse, and in a sense the generated architectures represent two orthogonal construction strategies: The first algorithm constructs a special kind of multi-layer feed-forward networks, where each hidden layer consists of just one single unit which receives a connection from each of the network's original input terminals as well as from the units in all the previous "one-unit" layers. We shall refer to this algorithm as the *High-Order Cascor Algorithm*, and it was developed by Fahlman and Lebiere in 1990 [Fahlman *et al.*]. The *Second-Order Cascor Algorithm*, on the other hand, is the algorithm that we have developed, and it uses the quite opposite strategy: it generates only a two-layer feed-forward network (with shortcut connections between the input terminals and the output units), i.e. it adds all new units to the same hidden layer. Section 4.2.1 and 4.2.2 below, present a thorough description of each of these two algorithms, while Section 4.3 is devoted to a comparison and discussion of their different properties in terms of hypothesis representation and hypothesis space evolution.

4 Two Dynamic Learning Algorithms

Since the late eighties, several network construction algorithms which use different schemes to generate various types of network architectures have been developed. In 1989 Mézard and Nadel [Mézard *et al.*] proposed the Tiling Algorithm which creates multi-layer architectures, and Ilsøe and Poulsen [Ilsøe *et al.*] presented the Minimum Algorithm which is based on a simple (dynamic) extension of the generalized delta rule. In 1990 an additional number of dynamic learning algorithms were presented, and perhaps the two most famous of these are the Upstart Algorithm by Frean [Frean], and the Cascade-Correlation Learning Architecture by Fahlman and Lebiere [Fahlman *et al.*]. In this chapter we introduce yet another network construction algorithm which we have developed with reference to this project. The algorithm is to a large extent similar to Fahlman and Lebiere's Cascade-Correlation Learning Architecture, as they both employ exactly the same learning procedure to train the networks; they differ, however, decisively in the way they build up these networks during the training process. We describe these two algorithms more thoroughly in Section 4.2 and 4.3, but first we want to elaborate a little more on function realization by multi-layer network structures, and to introduce the concept of feature detectors.

4.1 Feature Discovery

In Section 3.1 we defined the event space as the cross product of the sets of all possible feature values, which implies that any presented input example corresponds to a n-dimensional feature vector in the event space. In this sense, the task of an artificial neural network is simply to discover, extract, and evaluate the most important features (i.e. the important properties or attributes) that characterize the distinct classes. However, as we demonstrated in Chapter 2 and by the XOR example on page 27, it is only in certain (simple) cases that a network is able to properly extract and evaluate these characterizing features, using only one layer of units. For most applications a network has to contain at least one layer of hidden units, in order to realize the desired function. That is to say, the units in the output layer cannot in just one step correctly evaluate and compare all the distinct features; an intermediate layer of hidden units have to carry out a kind of preprocessing or preevaluation of the input features before the output units are able to accomplish the final classification. Thus the output from the entire hidden layer corresponds to a new set of features and, similar to the hidden units, the output units now have to discover, extract, and evaluate the most important ones. In this sense the units in the first layer can be characterized as *first-order feature detectors*, and the units in the second layer as *second-order feature detectors*. In Section 2.3.1 we described this idea formally, and we generalized it to any k-layer Perceptron by considering the function specified by the Perceptron as composed of k subfunctions, one for each layer.

From this point of view, any multi-layer network is thus capable of computing higher-order correlations, and there is therefore reason to believe that such networks will discover higher-order features and build up an internal representation which mirror a possible inherent hierarchic structure of the problem domain. Perhaps the most famous example of this, is NETtalk developed by Sejnowski and Rosenberg in 1987 [Sejnowski *et al.*].

$(1,1,1)$. This projection, however, enables the output unit to create a hyperplane L_2 which separates the two vertices represented by a $\bullet$ from the two vertices represented by a $\circ$, i.e. the vertices (0,1,0), (1,1,0), and (1,0,0) constitute a hyperplane cut of H^3, where (1,1,0) corresponds to an empty subset of the event space. The decision region S_2 of the output unit is defined by the bias $\theta_2 = \frac{1}{4}$ and the weight vector $w_2 = (\frac{1}{2}, \frac{1}{2}, -1)$, and we have $f_2(x_1, x_2, f_1(x_1, x_2)) = 1$ if $\frac{1}{2}x_1 + \frac{1}{2}x_2 - 1f_1(x_1, x_2) \geq \frac{1}{4}$.

Now, let us reconsider the discussion of generalization in a dynamically expanding hypothesis space. In Figure 14 it is easy to see how the hypothesis representation becomes more powerful when the weight space is expanded due to the incorporation of the hidden unit, and hence how the network is enabled to deal with more complex hypotheses. However, the importance of *how* the hypothesis space is expanded should be emphasized. As we demonstrated in Chapter 2 and stressed again in the previous section, the computational power of an output unit is greater then the computational power of a hidden unit, as the information or data it is fed with has already been processed by the units in the hidden layer. That is to say that the input patterns presented to the output unit represent a kind of "intermediate results" or "temporary preclassification" of the original input patterns, which greatly enhance the output unit's ability to perform the final classification. By noticing how the number of layers becomes of importance for the networks' capability of realizing complex functions, the generalization of this to multi-layer feed-forward networks is obvious.

The number of first-layer hidden units, on the other hand, determines the number of hyperplanes in the event space, and as we demonstrated in Proposition 2 and Proposition 3 in Chapter 2, an insufficient number of such hyperplanes causes that the network will be unable to form a satisfactory decision region. It is therefore very important to be aware of how the hypothesis space is expanded. A network's ability to represent powerful hypotheses is *not* just a question of expanding the hypothesis space sufficiently. In the next chapter where we focus on two specific dynamic learning algorithms, we will discuss the importance of a powerful hypothesis representation more thoroughly.

Another important point which we believe should be emphasized, is the following. We know that the size and the contents (e.g. the amount of redundant information) of the applied training set have a major influence on the number of hidden units required for a network to realize a given function. We also know that the same is true for the network's ability to generalize. Therefore the configuration of a dynamic neural network will be determined by these properties, instead of the *intended* function itself. The configuration of a static neural network, on the other hand, is often based on a rough guess on the complexity of the particular function or mapping it is intended to realize. That is to say that the configuration of a dynamic neural network, unlike a static neural network, is determined by the *composition* of the applied training set by repeatedly measuring the actual performance on the different training examples and then letting this be decisive for the necessary number of hidden units. In this way, the hypothesis representation will only be as complex as the given training set for the particular application requires, and thus, in the same sense as before, the network strives to discover the simplest possible generalization.

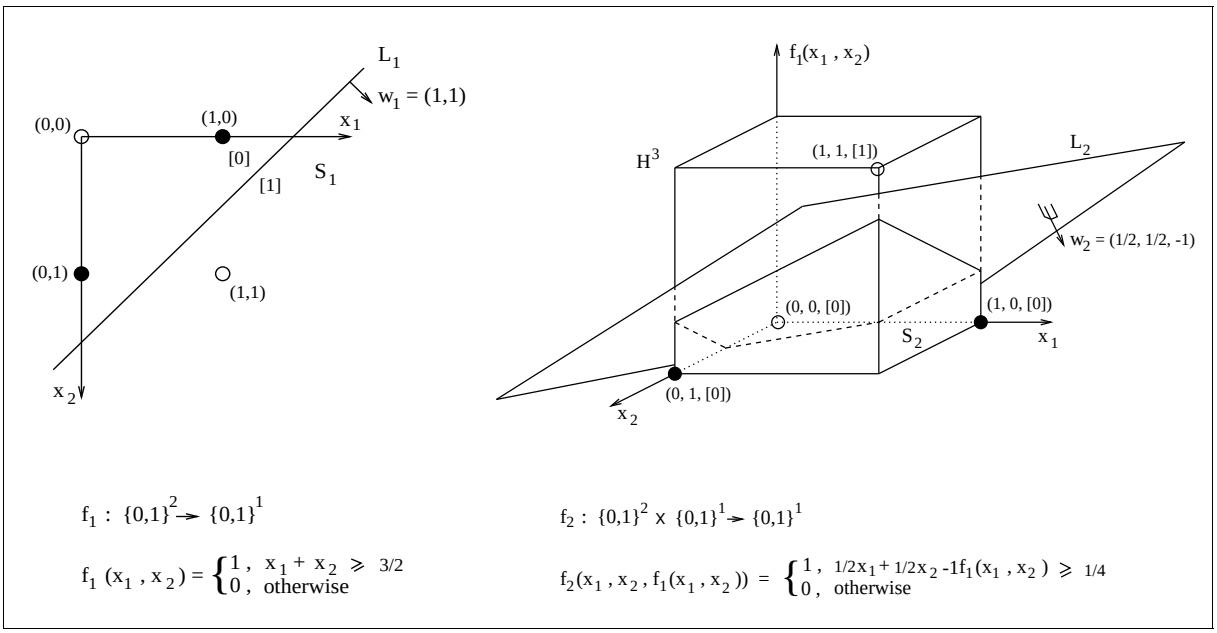

Figure 14: Geometric illustration of the decision regions of the hidden unit (to the left) and the output unit (to the right) for a network realizing the XOR function. See the example for explanation.

with one output unit, which we have analyzed thoroughly in Section 2.2 and 2.3. The truth table for the XOR function is illustrated geometrically to the left in Figure 14 (a $\bullet$ ($\circ$) represents a desired output of 1 (0), respectively), and its is easily seen that a single-layer Perceptron is unable to correctly classify all the four training instances as they are not linearly separable. That is to say that the hypothesis space is too simple or restricted to contain a valid generalization, and the algorithm expands therefore the search space by adding a hidden unit to the network. It turns out that this network[18] is capable of realizing the XOR function, and a possible solution found by the algorithm is shown to the right in Figure 13.

In Figure 14 we have used the framework outlined in Chapter 2 to illustrate the found solution geometrically by considering the decision region of the hidden unit and the output unit, respectively. The bias value $\theta_1 = \frac{3}{2}$ and the weight vector $w_1 = (1,1)$ determine the hyperplane $L_1 \subseteq I\!R^2$ for the hidden unit, and in Figure 14 it is easily seen that this unit computes the Boolean AND function, i.e. $f_1(x_1, x_2) = 1$ if $1x_1 + 1x_2 \geq \frac{3}{2}$, and 0 otherwise. Due to the special configuration of the network, we notice that the function f_2 computed by the output unit not only is a function of f_1 (which is the case for two-layer Perceptrons as we demonstrated in Section 2.3.1), but also of the two original input values x_1 and x_2: $f_2(x_1, x_2, f_1)$. That is to say, the decision region of the output unit is a subspace of the three-dimensional hypercube H^3 which is defined by the cross product of the original two-dimensional input space (or event space) $\{0,1\}^2$ and the one-dimensional classification space $\{0,1\}^1$ determined by f_1. This is illustrated to the right in Figure 14, and we observe that the AND function f_1 only affects the single training instance $(1,1) \in S_1 \subseteq I\!R^2$ by projecting it onto the hypercube vertex

[18]Note that the network does not belong to the class of two-layer Perceptrons discussed in Section 2.3, but is restricted to the more general class of feed-forward networks.

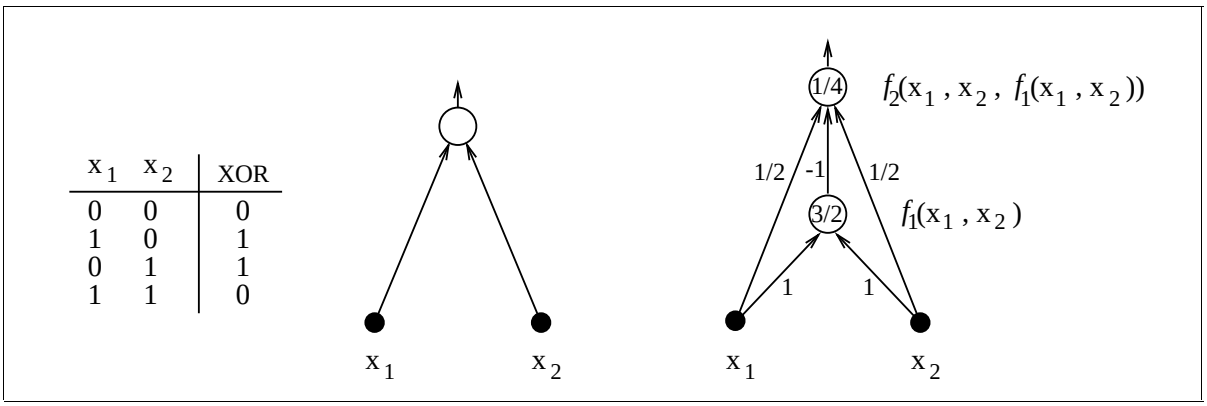

Figure 13: Two networks considered by a dynamic learning algorithm when teaching the Boolean XOR function. The network on the left is the initial smallest possible network determined exclusively by the chosen representation of the environment, i.e the number of inputs and outputs in the truth table. This single-layer Perceptron is too simple to realize the function, and therefore the learning algorithm adds a hidden unit to the network. This expanded network is the smallest possible network which is capable of realizing the XOR function, and a possible solution found by the algorithm is shown on the right. Notice that the output function f_2 is a function of f_1 i.e. the function computed by the hidden unit, as well as the two original input values.

algorithm will typically start to train the smallest possible network which *might* be able to realize the desired function, i.e. a network with input and output units only, which encodes the environment according to the chosen representation. (See Figure 13 and the example in Section 3.3.1 below.) If this single-layer network is unable to realize the function, a hidden unit is added to the network and learning is resumed. If this network too is unable to realize the function, yet another hidden unit is added and so on, until (ideally) the training set has been learned and a valid generalization is found. However, each time a new hidden unit is added to the network, the hypothesis space is expanded. That is to say that the hypothesis representation gradually becomes more powerful, and the network is therefore enabled to deal with incrementally more complex hypotheses, until the "simplest possible" generalization has been discovered. Simplest possible should not be taken literally; of course we are not guaranteed to find the simplest of *all* possible hypotheses. What we mean, however, is that given the sequence of various local minima that are considered during a particular training session, the expansion of the hypothesis space ends when the first hypothesis which is consistent with the training instances has been discovered. That is, the hypothesis representation will not be more complex than necessary. This means also that the number of degrees of freedom of the network will be minimal, which we have argued for being advantageous for the network's ability to generalize. Let us illustrate this dynamic evolution of a network more thoroughly by considering a simple example.

3.3.1 Example: The Boolean XOR Function

The two networks considered by a dynamic learning algorithm teaching the XOR function are shown in Figure 13. The initial smallest possible network is a single-layer Perceptron

network the number of weights, i.e. the dimension of the hypothesis space, has to be laid down before the actual learning starts, and thereafter remain fixed. However, if the cardinality of the search space turns out to be inappropriate for the task considered, the network will either generalize poorly (too many weights), or it will never be able to learn the mapping exemplified in the training set (too few weights). One way to overcome this major disadvantage to such networks, is in some sense to ignore the problem, and instead leave it to the applied learning algorithm to configure the network. This is the subject of the next section.

3.3 Generalization in Dynamic Neural Networks

In the previous section we demonstrated that the network architecture is of crucial importance for a network's ability to generalize, and that each application requires its own architecture. We also mentioned that a network's ability to realize or learn a certain function and its ability to generalize, are two different properties attached to the network. Furthermore we argued that to obtain a good generalization one has to be aware of distinct types of biases and see through these, in order to "build into the network" as much knowledge about the problem as possible, and to appropriately limit the number of weights. It is therefore desirable to employ a learning algorithm that not only finds the optimal set of weights for a given architecture and a given application, but also optimizes the architecture itself with special reference to the networks ability to generalize. One class of such learning algorithms are dynamic learning algorithms which during the teaching of the net, repeatedly construct or modify the architecture to suit the particular problem considered.

Basically, there are two different methods for dynamic configuration of an artificial neural network[17] when we want to limit the number of weights (or the number of degrees of freedom): Either to start with an excessive number of hidden units and then remove the most unimportant connections or units (pruning), or to start with an insufficient network capacity and then gradually increase the number of hidden units (network construction). In the rest of this thesis we shall entirely focus on the latter of these, and the basic reason for this is optimization. It seems rather silly for the pruning methods first to use a lot of cpu power to properly adjust a possible large number of unnecessary weights, and then afterwards to use additional cpu power to find and eliminate the most unnecessary ones. That is to say that we first use a lot of effort to search for a good generalization in an unnecessary large version space (with the compounded possibility of misclassification and "overfitting"), and then afterwards "modify" the found generalization by reducing the dimension of the version space.

Most network construction algorithms, on the other hand, will in the light of this definitely be more efficient. Furthermore, we believe that the search for a good generalization in a neural network, under the assumption that constraints are a necessity to effect generalization, with advantage could be interpreted in the context of a dynamically expanding hypothesis space. Let us substantiate this issue more thoroughly. A network construction

[17]Actually there is a third method, namely to apply genetic algorithms to perform a search among all possible (trained) network architectures. This approach, however, is beyond the scope of this thesis.

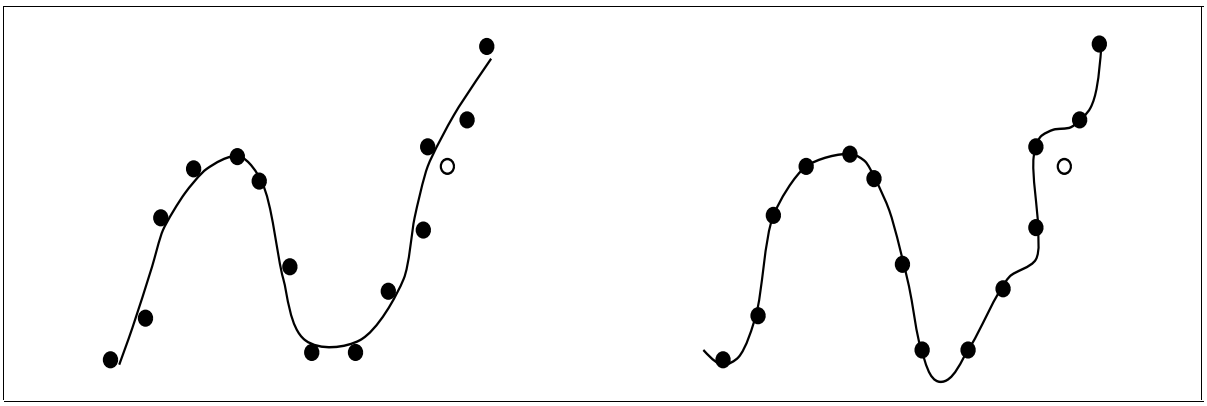

Figure 12: To the left an example of a good fit to noisy data. To the right, overfitting of the same data. The "training" data have been learned perfectly, but on a test example (○) the fit is poor.

the previous section and we found that a restricted hypothesis representation (such as using only one rectangle) could prevent certain generalizations from ever being considered, while a more powerful representation would enable the learner to deal with more complex hypotheses. However, *the more powerful the representation is, the larger is the version space, and consequently the possibility of misclassification will increase.* In the context of artificial neural networks that is to say that the number of hidden units (or actually weights since the number of weights determines the dimension of the version space) is of crucial importance for a network's ability to generalize, and that too many hidden units probably will result in a poor generalization ability. This coherence has in fact been demonstrated empirically in many experiments [Jakobsen *et al.*], [Hertz *et al.*], [Honavar *et al.*], [Sjøgaard], and as we have mentioned before, a large volume of research has been (and still is) devoted to the problems of network design. One of the widely known rules of thumb for good generalization is that the number of examples in the applied training set should be several times larger than the number of degrees of freedom in the network, i.e. the number weights. This can be understood intuitively by noting that if the number of degrees of freedom in a network is larger than the number of constraints[16] imposed by the training instances, the applied learning algorithm will be unable to completely constrain all the weights in the network. This allows effects of initial conditions to interfere with the learned information, and the trained network's ability to generalize may degrade.

Another important way to focus on generalization in neural networks is from a statistical point of view. Consider the task of curve-fitting illustrated in Figure 12. As illustrated to the right, a curve fitted with too many parameters follows all the irrelevant details (e.g. noise) and is very poor for interpolation and extrapolation; too many free parameters result in overfitting. The same is true for neural networks: too many weights give a poor generalization.

The reason why we have used the term *static neural networks* is due to the fixation of the hypothesis space during the search for a good generalization. In a static neural

[16]In [Widrow *et al.*] the number of constraints is defined to be $N_y N_p$, where N_y is the number of output units in the network and N_p is the number of patterns in the training set.

the reduction of the version space proceeds differently. Initially, we start with a very large version space due to the random initialization of the weights. This initial version space contains already the hypothesis that we will end up with[15]; the learning task is "only" to repeatedly discover better and better hypotheses until we find one which is "consistent" with the training instances, i.e. until the average error is reduced to a satisfactory level. And by using the error function as an inverse utility function for each hypothesis, we obtain a total ordering of all possible hypotheses which makes a direct comparison of any two hypotheses very simple. Furthermore, if we use a gradient descent learning algorithm, we will now automatically reduce the size of the version space after each weight update by discarding all the hypotheses that exhibit a smaller utility than the current best hypothesis. That is to say that we never *alter* the current best hypothesis; instead we find a new hypothesis which has a higher utility, and make that the new current best hypothesis. And as the average error gradually decreases, we are guaranteed that more and more training instances automatically are correctly classified by the sequence of current best hypotheses discovered. So, in this sense we don't have to perform any direct consistency check at all! The consistency of the final hypothesis discovered, is automatically ensured by the monotonically decreasing error function. (This implies, of course, that we don't get stuck in a local minimum and thus that the error can be reduced to a satisfactory level.)

3.2.1 Biases

According to the discussion above, a gradient descent learning algorithm e.g., is now guaranteed to find a generalization. But depending on what could be called *algorithmic biases*, some generalizations will be more likely than others. Mitchell [Mitchell80] has defined a bias as "any basis for choosing one generalization over another, other than strict consistency with the observed training instances." In this sense the initial weight values are algorithmic biases which inevitably make some hypotheses more preferable than others as the algorithm will start its search for a good generalization at that point in the hypothesis space.

However, according to Mitchell's definition, there exists in the context of artificial neural networks at least three additional types of biases, namely those arising from the chosen representation of the hypotheses, from the representation of the event space, and from the domain itself. *Domain biases* may be described as biases which cannot be controlled by the learning algorithm. A skewed probability distribution of the examples in the applied training set e.g., where certain types have a much higher frequency than others, may affect the search in the hypothesis space and thereby the generalization that the network ends up with. Similarly, an *event space representation* which is defective or insufficient would cause that the network would not be able to consider certain generalizations.

Finally the importance of the *hypothesis representation* should be emphasized, and in fact this issue is the major subject of the next two chapters where we empirically compare and investigate two dynamic learning algorithms. We also discussed this issue in

[15]This implies that the network architecture remain fixed during the training process. In the next section where we discuss generalization in dynamic neural networks, the situation is more complicated as the hypothesis space –and thereby the version space– is expanded during the training process.

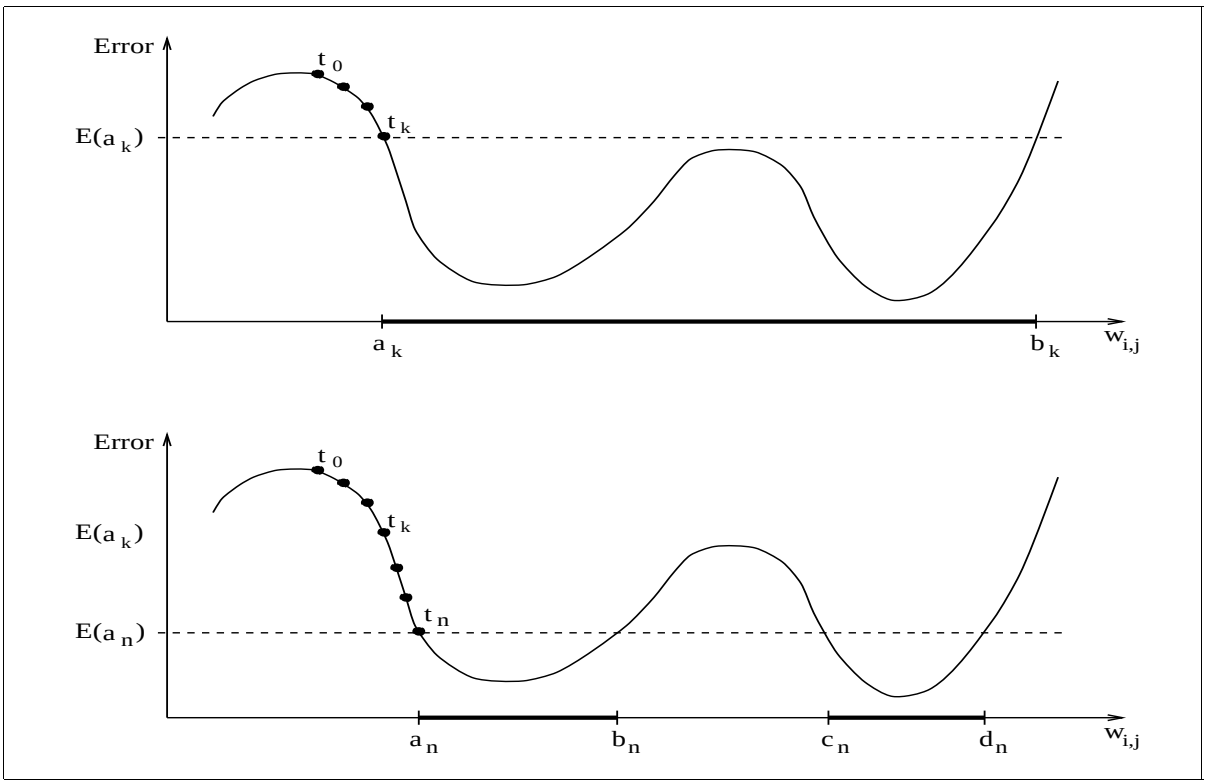

Figure 11: Geometric illustration of the version space of a network at time t_k (the upper part) and at time t_n (the lower part). At time t_k the version space is comprised by the set of hypotheses whose utility is at least $1/E(a_k)$, and at time t_n it is comprised by the set of hypotheses having a utility greater than $1/E(a_n)$. See the text for further explanation.

$1/E(a_k)$, i.e. the version space is comprised by the set of hypotheses that imply that $w_{i,j} \in [a_k, b_k]$.

The situation at time t_n $(t_n > t_k)$ is depicted in the lower part of Figure 11. Now $w_{i,j} = a_n$ and since $E(a_n) < E(a_k)$, the utility of the current best hypothesis is higher than the utility of the current best hypothesis at time t_k. And, similarly to the situation at time t_k, the version space is comprised by the set of hypotheses that imply that $w_{i,j} \in [a_n, b_n] \cup [c_n, d_n]$, and since $[a_n, b_n] \cup [c_n, d_n] \subset [a_k, b_k]$ the version space at time t_n is smaller than the version space at time t_k.

The generalization of this to the case where all the weights are changed simultaneously during the training process is obvious, and if we use a gradient descent learning algorithm we are guaranteed that the size of the version space will be reduced after each weight update.

Now, let us try to summarize and stress how the version space is gradually reduced in the two different learning systems. In the context of Machine Learning, the data-driven generalization strategies prescribe that the current best hypothesis is altered during the learning process, whereupon a very costly consistency check with the past training instances has to be performed. That is, the version space changes each time a new training instance is presented to the learner. In the context of artificial neural networks,

major drawback of the corresponding data-driven generalization strategies, as tests and comparisons of the hypotheses then are very costly in computational power.

The definition of consistency is very strict because it prescribes that *all* the successively derived current best hypotheses are one hundred percent valid generalizations. The utility concept, on the other hand, operates on all the training instances simultaneously, and therefore we can in no way expect that (all) the hypotheses derived during the learning process will be one hundred percent valid generalizations. Informally, we can characterize the utility concept and the consistency concept as follows:

- By using the utility concept, we can easily measure how much better one hypothesis is compared to another, i.e. we have a metric or measurement which indicates "how good" the current best hypothesis is, relative to *all* the training instances.
- The consistency concept, on the other hand, does not require such a measurement since, by definition, the current best hypothesis always is one hundred percent valid. The validity, however, counts only for a subset of the training instances.

The back-propagation learning algorithm, or any other algorithm which performs a gradient descent search in the weight space (i.e. the hypothesis space), will gradually discover better and better hypotheses until a local minimum is reached: For each iteration, the algorithm takes a little step downhill in the error surface in order to minimize the error function. This means that the utility of the (temporary) hypotheses considered during the learning process will increase monotonically until the best (local) hypothesis has been discovered. And in this sense, artificial neural networks overcome the awkward drawbacks of maintaining consistency during the search for a valid generalization, and we shall therefore define that the various hypotheses which have a higher utility than the current best hypothesis, comprise the version space.

We have tried to illustrate this definition geometrically in Figure 11. Given a network $\mathcal{N}$ with fixed architecture and a particular training set $\mathcal{D}$, the error function, $E_{\text{LMS}}^{\mathcal{N}\mathcal{D}}$ say, is a function of just one parameter, namely the weight vector $\mathbf{w}$:

$$\begin{aligned} E_{\text{LMS}}^{\mathcal{N}\mathcal{D}}(\mathbf{w}) &= \frac{1}{2}\sum_{p=1}^{N_{pat}}\sum_{i=1}^{N_{out}}(o_i^p - t_i^p)^2 \\ &= \frac{1}{2}\sum_{p=1}^{N_{pat}}\sum_{i=1}^{N_{out}}(out_i^p(\sum_j w_{i,j}o_j^p) - t_i^p)^2 \end{aligned}$$

where N_{pat} is the number of training patterns in $\mathcal{D}$, and N_{out} is the number of output units in $\mathcal{N}$. (See the introduction to artificial neural networks in Section 2.1.) In order to illustrate how the size of the version space is reduced during the training process, we have depicted $E_{\text{LMS}}^{\mathcal{N}\mathcal{D}}$ as a function of just one weight $w_{i,j}$ (while keeping all other weights fixed) in Figure 11. (To keep things simple, we have omitted the indices on the error function E.) At time t_k, $w_{i,j}$ has the value a_k and the error is therefore $E(a_k)$, i.e. the utility of the current best hypothesis is $1/E(a_k)$. (See the upper part of Figure 11.) The version space is therefore comprised by the set of hypotheses whose utility is at least

3.2 Generalization in Static Neural Networks

In this section we argue that the framework presented above can be applied to the context of artificial neural networks. This approach, however, requires a few minor adjustments which we shall discuss first. Then we demonstrate how elegantly neural networks overcome the awkward drawbacks discussed above, and we outline, in terms of the adapted framework, the basic ideas of applying dynamic learning algorithms to train artificial neural networks.

First, let us demonstrate how the framework should be interpreted in the context of artificial neural networks. Clearly, the n inputs to a network define the event space as any input will consist of a n-dimensional vector of feature values. Similarly, the weight space constitutes the hypothesis space. To see this, consider a single-layer Perceptron with one output unit as defined in Section 2.2. This network uses a hyperplane in the event space to classify all the presented input examples into one of two classes. The corresponding positive half-space, which we denoted as the decision region, determines the function being computed, and it represents therefore a hypothesis. And since the orientation and position of the hyperplane is defined solely by the weight values, the weight space constitutes the space of all possible hypotheses. This imply that the process of deriving a rule or hypothesis simply corresponds to an adjustment of the weights which ensures that all the positive training instances are in the decision region of the output unit. So, indeed learned generalizations correspond to different subspaces of the event space, constrained by the set of observed training instances. It is now easy to see that this could be generalized to arbitrary networks, but clearly, having the geometric presentation in Section 2.3 and Figure 10 in mind, the number of units and layers determines the possible representations used for describing the hypotheses. The number of first-layer units in a multi-layer Perceptron e.g., determines the number of hyperplanes in the event space, while the number of layers becomes of importance for the network's capability of realizing complex functions.

The determination of the version space is a little more tricky. The reason is that most learning algorithms use an average error measure to adjust the weights, i.e. each single training instance is not learned rigorously one by one. Instead all the training instances are learned "simultaneously," and the error is an average measure of how well the network performs on the entire training set. This means that the definition of consistency will not be valid in the context of artificial neural networks, and we have to come up with something else in order to compare and test the distinct hypotheses derived during the learning process. And for this we can use the error function[14] which we can think of as an inverse utility function for each hypothesis. In this sense we shall define that one hypothesis, H_1 say, is preferable to another, H_2, if the error at the point in weight space corresponding to H_1 is less than the error at the point corresponding to H_2, i.e. the utility of H_1 is larger than the utility of H_2. Now we have obtained a total ordering of all possible hypotheses which makes a direct comparison of any two hypotheses very simple. In the context of Machine Learning, the strict consistency concept only enables a partial ordering of the hypotheses discovered, and as mentioned before that makes up the

[14] For example the least mean square function E_{LMS} which sums the square error over all output units i and all training patterns p: $E_{\mathrm{LMS}} = \frac{1}{2}\sum_{p=1}^{k}\sum_{i=1}^{l}(o_i^p - t_i^p)^2$

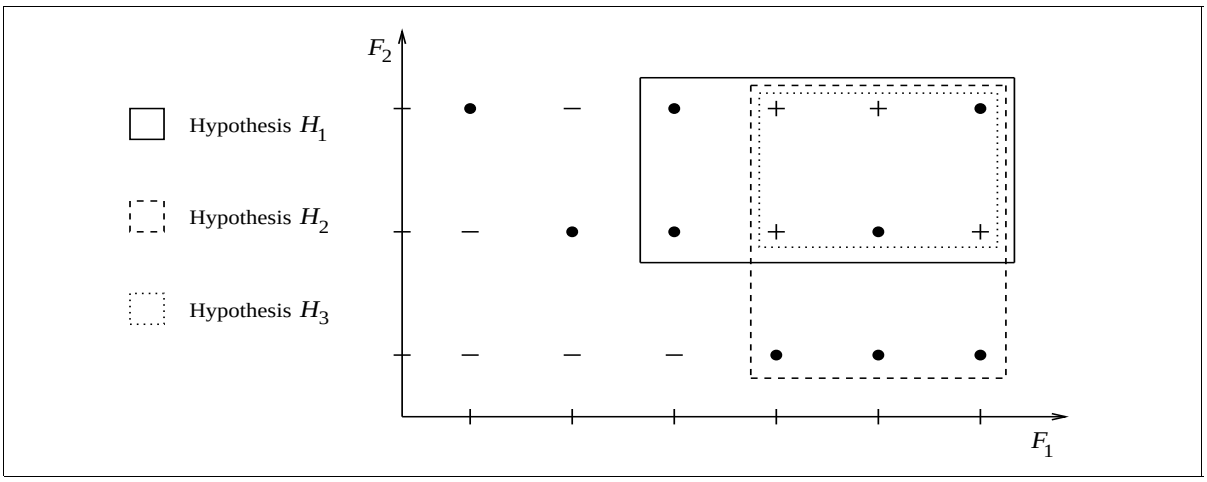

Figure 10: A two dimensional event space E defined by the two (discrete) features F_1 (with six possible values) and F_2 (with three possible values). $+$ $(-)$ represent positive (negative) events, respectively, and until now only these nine events have been presented to the learner. H_1, H_2, and H_3 are the only consistent hypotheses left at this time, and H_3 is the most specific.

concepts, but the size of the version space would increase and so would the probability of misclassification.

As mentioned above, it is usually infeasible in practice to present the learner with enough examples to exhaust the version space. But as more and more examples are seen, the generalizations that are produced will be increasingly accurate with respect to the true hypothesis, and fortunately it is often the case that the learner only has to come up with a good approximation to the correct hypothesis. The greater the allowed tolerance is, the fewer training examples are needed.

Now, let us summarize these considerations. The hypothesis space of possible solutions consists of the set of all possible hypotheses representable by the learner. The version space is a subset of the hypothesis space and whenever a certain number of training instances have been presented to the learner, the version space consists exclusively of the remaining set of consistent hypotheses. That is to say that the hypothesis space is fixed during the learning process, while the version space shrinks as the set of consistent hypotheses will be reduced when the number of presented examples increases. But whenever the current best hypothesis[13] is altered during the learning process, it is very costly to test each alteration for consistency with the past training instances. Some of the generalization strategies described by Mitchell use a backtracking technique to reconsider previously processed training instances. Other strategies involve a direct comparison of generalizations in order to determine whether one is more general than another. The point is, however, that such tests and comparisons make up the major drawback of these data-driven generalization strategies as they are very costly in computational power. Indeed such an *ad hoc* maintenance of consistency during the generalization process is inefficient.

[13]or, as before, a set of alternative consistent hypotheses

tion of a class of objects (or concept learning) given a set of representative examples, as a search problem [Mitchell82].

It will be assumed that the input examples (or instances) are described by a finite number of features, each of which can assume one of a (possibly infinite) set of values. That is, each instance corresponds to a feature vector in an *event space* E, defined as the cross product of the sets of all possible feature values. The applied training set is then a collection of a finite number of unordered instances in E. Learned generalizations correspond to sets of these instances in the event space, and we can think of the process of deriving a rule or hypothesis as assigning different subspaces of E to the various classes. The set of all hypotheses representable by the learner forms a *hypothesis space* (or search space) of possible solutions, and "the learning task is to examine this hypothesis space, subject to constraints imposed by the training instances, to determine plausible generalizations" [Mitchell82, page 205]. Usually the search is restricted to the set of all consistent hypotheses, which Mitchell denoted as the *version space*. A hypothesis is consistent if it does not contradict any of the observed instances, and in the following we shall assume that learning proceeds with consistent training instances only.

Having defined this framework, let us now focus on the process of generalization or the search for valid generalizations. We shall not go into depth about the distinct data-driven search strategies discussed by Mitchell, but only use the framework to outline the basic ideas. The most fundamental thing to notice is that the training instances are presented to the learner one by one. That is, iteratively a new instance is selected and removed from the training set and tested against the current best hypothesis[12]. This hypothesis is then altered as needed so that the resulting generalization is consistent with this new instance as well as all the previous instances. Then a new instance is selected and so on, until all the training instances have been presented to the learner. This corresponds to that the version space will shrink as the number of presented examples grows. When, or if, the version space contains at most one hypothesis, it is said to be exhausted, and only then the learner can be confident about the validity of a derived rule. However, most often the version space will never be exhausted and a number of valid hypotheses will exist. The actual number of such consistent hypotheses at a given time will depend on three things, namely the representation used for describing the event space, the representation used for describing hypotheses, and, of course, the number of presented training instances. To illustrate this consider Figure 10, and suppose that the learner uses a rectangle to classify the positive events in E. Then H_1, H_2, and H_3 are all perfectly valid generalizations. Each of them include all the positive events and exclude all the negative ones. However, one of them may be discarded when the next event is considered as it is possible that this event will contradict it. H_3 is the most specific hypothesis which is consistent with the observed instances but perhaps it is too specific, while H_1 and H_2, on the other hand, may be too general. It is impossible to select the right one of these without seeing at least one more example. If the features F_1 and F_2 were continuous there would still remain infinitely many consistent hypotheses, so clearly the representation of the event space is important for the learner to come up with a useful generalization. And similarly, if the learner was using two rectangles instead of one, it could represent more complex

[12]Actually there may be a whole set of several alternative hypotheses depending on the particular search strategy considered.

3 The Process of Generalization

Generalization has been studied on many different contexts, and in the areas of e.g. Pattern Recognition, Linguistics, and Machine Learning, generalization is definitely one of the key issues. The basic paradigm we shall consider here is supervised learning from examples, which in the context of artificial neural networks could be described as follows: A teacher presents a network with a set of training examples (or events) partitioned into a number of classes. The examples are propagated through the network and the output signals are compared with the corresponding known correct answers. If there is any inconsistency between the actual and the desired output, the network receives a feedback about the error. The task is thus to "teach" the network to produce the correct answers to the presented examples in the training set. However, for most applications we certainly want more than that. We want the network to be able to generalize, i.e. to construct a rule or hypothesis consistent with the training set, which enables the network to correctly classify future events. The fundamental difficulty is, however, that given a small number of training examples, there may exist a large number of consistent hypotheses, and *a priori* there is no basis for choosing one over the other.

Inspired by traditional Machine Learning and the famous paper *Generalization as Search* by Tom Mitchell [Mitchell82], we propose in this chapter a different conceptual approach to generalization (or actually to the process of generalization) in artificial neural networks. The two most important ideas are (1) to consider the problem of forming a general description of a class of objects as a search problem, and (2) to divide the search space into a static and a dynamic part. These ideas are beneficial as they emphasize the evolution or process that the learner must undergo in order to discover a valid generalization. We find that this approach and the adapted conceptual framework provide a more varied and intuitively appealing view on generalization.

In Section 3.1 below, we describe the basic ideas of Mitchell's work and we define the corresponding conceptual framework. Furthermore we focus on a number of interdependent factors which affect the generalization process, and which therefore should be given some consideration. In the two succeeding sections, we argue for the advantages of applying the conceptual framework in the context of artificial neural networks: Section 3.2 concerns the adaptation of the framework for artificial neural networks in general, and a number of important issues are carefully discussed. Section 3.3 goes one step further and discusses the advantages of applying the framework on dynamic neural networks, and a simple example is presented. Besides, having the general analysis of feed-forward networks in the previous chapter in mind, the example demonstrates how the framework can be used to explain the evolution that dynamic neural networks undergo during the training process.

3.1 A Conceptual Approach to Generalization

In this section we shall temporarily forget the context of artificial neural networks and instead focus on learning and generalization from the somewhat broader perspective of Machine Learning. Basically we shall consider the problem of forming a general descrip-

time consuming and indeed it would be profitable if one could come up with an initial near-optimal guess on such an architecture. That is, however, difficult as the necessary *a priori* knowledge depends on several interdependent factors as for instance:

- The complexity of the function to be realized: A complex function usually demands more layers and units than a simple function. But how do we measure the complexity of a particular function?
- The representation of the environment: E.g., should each class be represented by a specific output unit or would a binary coding of the classes be more appropriate?
- Noise: Does the function exemplified in the training set contain disturbing noise?
- The size of the training set: Usually a "large" training set demands more hidden units than a "small" training set. What is "large" and "small" for the particular problem considered?
- The content of the training set: A large training set which contains a lot of redundant information is not necessarily better than a smaller training set without redundant information. Does the particular training set contain verbose and unnecessary information?

One way to overcome most of these difficulties is simply to ignore them and instead leave it to the applied learning algorithm to take them into account. And this is one of the main reasons why we shall investigate the capabilities of two *dynamic* learning algorithms in this thesis. However, before we discuss this issue any further we should focus on one of the most important reasons to consider neural networks at all, namely generalization. Without generalization neural networks would not be very interesting. This is the topic of the next chapter.

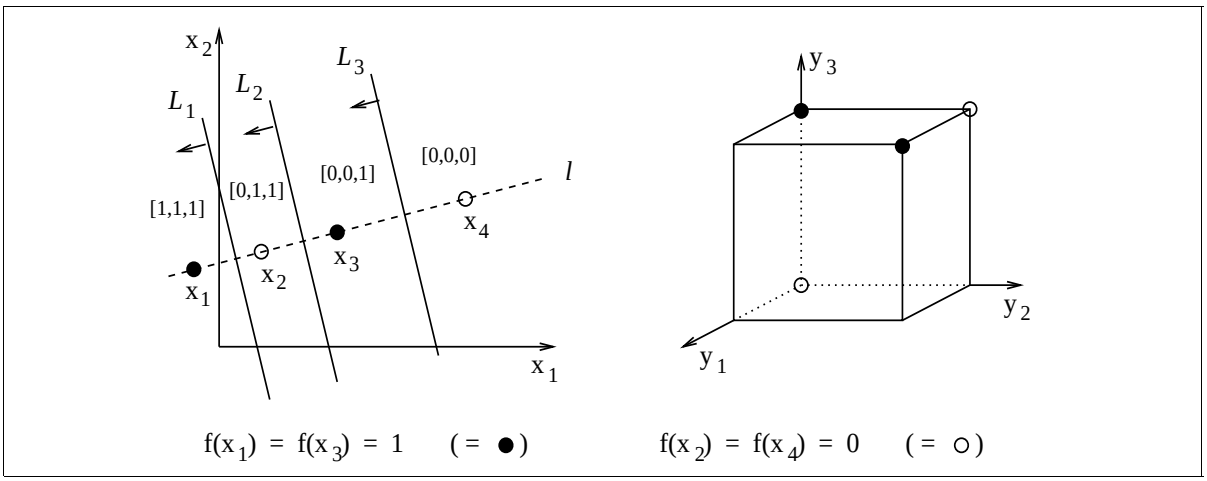

Figure 9: Geometric illustration of the properties of the function f. The points $\mathbf{x}_1, \cdots, \mathbf{x}_4$ lie in succession on the dashed line l. See text for explanation.

In terms of neural nets and function realization this may be "translated" into the following: Let $\mathcal{F}$ be a class of functions on the form $f : I\!R^n \mapsto \{0,1\}$ and let $\mathcal{D}$ be a m-element (training) set in $I\!R^n$. A *dichotomy* induced on $\mathcal{D}$ by $f \in \mathcal{F}$ is a shattering of $\mathcal{D}$ in disjoint subsets $\mathcal{D}^+ = \{\mathbf{x} \in \mathcal{D} \mid f(\mathbf{x}) = 1\}$ and $\mathcal{D}^- = \{\mathbf{x} \in \mathcal{D} \mid f(\mathbf{x}) = 0\}$. That is, for a given f, $\mathcal{D}^+$ is the decision region S of f. Let $\Delta_{\mathcal{F}}(\mathcal{D})$ denote the number of dichotomies that can be induced on $\mathcal{D}$ by functions $f \in \mathcal{F}$. Then we have $\Delta_{\mathcal{F}}(\mathcal{D}) \leq 2^{|\mathcal{D}|} = 2^m$. The V-C dimension of a class of functions $\mathcal{F}$ is defined to be the cardinality of the largest, finite set $\mathcal{D} \subseteq I\!R^n$, that can be shattered by $\mathcal{F}$.

It has been clearly demonstrated in [Jakobsen *et al.*] that this measure of the required number of examples in the training set to learn a network to compute a given function, is much too conservative to have any practical importance. On a simple geometric classification problem using a two-layer Perceptron with six hidden units, Jacobsen *et al.* found that the theoretical number of examples in the training set should be 100.000, whereas their empirical results showed that 100 examples were sufficient.

We shall not go any further into a discussion of the theoretical importance of the V-C dimension in this thesis. The reason why we have mentioned it, however, is that we want to emphasize that there (still) is a huge gap between the results obtained theoretically on the one hand, and empirically on the other regarding network design and function realization. The reason for this is that the theoretical results, including Proposition 4, are built upon assumptions which are too general from a practical point of view. Both Proposition 4 and the definition of the V-C dimension focus on an entire class of functions to be realized by a particular network, whereas a real-world problem usually corresponds to only one or a few instances of such a class. And indeed it is rare that a particular function which should be realized by a neural network happens to be "the worst-case" function as e.g. the one illustrated in Figure 9. From a practical point of view the utility of Proposition 4 and of the conception of the V-C dimension is therefore limited, and most often the design of a network has to proceed on an *ad hoc* and empirical basis. This means that one has to do a lot of experiments in order to find an appropriate network architecture tailored for the particular problem considered. Such experiments are very

Proof: As the proof presented in [Huang/Huang] is rather analytical we shall omit the formal details here in order to preserve the clarity of the presentation[9]. Instead we will give a geometric proof in terms of the topological properties we have investigated so far. The idea is to devote a unit to each of the first $k-1$ elements (training instances) in D (the training set). The decision boundary of the first unit should separate $\mathbf{x}_1$ from $D^1 = D - \{\mathbf{x}_1\}$, such that $\mathbf{x}_1$ belongs to the decision region of the first unit[10]. The decision boundary of the second unit should separate $\mathbf{x}_2$ from $D^2 = D^1 - \{\mathbf{x}_2\}$, and similar for $\mathbf{x}_3, \cdots, \mathbf{x}_{k-1}$. Now we have obtained a one-to-one correspondence between the k elements in D and k "special" vertices of the hypercube H^{k-1}, where the origin corresponds to the remaining k'th element. These k vertices of H^{k-1} are special in that way that *any* subset of these can be separated from the remaining vertices by a single hyperplane cut[11]. That is to say, the single output unit in $\mathcal{P}^{(2)}$ is now capable of forming any desired decision region of $\mathcal{P}^{(2)}$, in terms of the k "special" vertices of H^{k-1}.

Now, to prove that $m = k-1$ is the least upper bound on the number of hidden units, we specify a function $h : \beta \mapsto I\!N$, where every element in β is a doublet consisting of a particular k-element set $D \subseteq I\!R^n$ and a function $f : D \mapsto \{0,1\}$. Define $h(D, f)$ to be the minimum number of hidden units in $\mathcal{P}^{(2)}$ needed to realize the function f defined on D. Then, clearly, $h(D, f) \leq k-1$. To see that

$$\sup_{D,f} h(D, f) = k-1$$

consider the following example: Given a k-element set D, where $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k$ lie in succession on a line in $I\!R^n$, let f be a function defined on D such that $f(\mathbf{x}_i) \neq f(\mathbf{x}_{i+1})$, for any $1 \leq i \leq k-1$, i.e. no two neighbor points may produce the same output. (See Figure 9.) To realize f, at least $k-1$ hyperplanes are needed to separate these k elements from one another, as any hyperplane in $I\!R^n$ that does not contain the line l can only have at most one intersection with this line. That is to say that no hyperplane can separate two elements, say $\mathbf{x}_i$ and $\mathbf{x}_j$, $i < j$, where $f(\mathbf{x}_i) = f(\mathbf{x}_j)$ without separating at least a third element, say $\mathbf{x}_r$, where $f(\mathbf{x}_r) \neq f(\mathbf{x}_i)$ and $i < r < j$. So, $k-1$ units are needed to realize f. □

The problem of network design has been the subject of a large number of papers since the introduction of learning algorithms which could deal with multi-layer networks. In 1986 [Haussler] introduced the *Vapnik-Chervonenkis dimension* (V-C dimension), defined with respect to a concept class, to derive upper bounds, similar to Proposition 4, on the number of examples required to learn a concept within that class: Given a concept class $\mathcal{F}$ and a set of events $\mathcal{D}$, $\mathcal{D}$ is shattered by $\mathcal{F}$ if $2^{\mathcal{D}} \subseteq \mathcal{F}$, i.e. the power set of $\mathcal{D}$ is contained in $\mathcal{F}$. Intuitively, $\mathcal{D}$ is shattered by $\mathcal{F}$ if *any* classification of the events in it can be represented by $\mathcal{F}$. The Vapnik-Chervonenkis dimension of $\mathcal{F}$ is the size of the largest set of events that can be shattered by $\mathcal{F}$.

[9] The algebraic proof can be found in [Huang/Huang, pages 48–49]. In the same paper the result is extended to hold good for arbitrary real vector-valued functions $g : D \mapsto I\!R^l$, $l \geq 1$.

[10] Note that the enumeration of the k elements could not be arbitrary. But it can easily be proven that for any nonempty, finite set $D \subseteq I\!R^n$, there exists at least one element $\mathbf{x}' \in D$ which could be separated from the remaining elements in D.

[11] The reason, which we shall not prove here, is that these k vertices $\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_k$ are affine independent in $I\!R^{k-1}$, i.e. $\mathbf{v}_1 - \mathbf{v}_k, \mathbf{v}_2 - \mathbf{v}_k, \cdots, \mathbf{v}_{k-1} - \mathbf{v}_k$ are linearly independent and therefore they constitute a basis which spans $I\!R^{k-1}$.

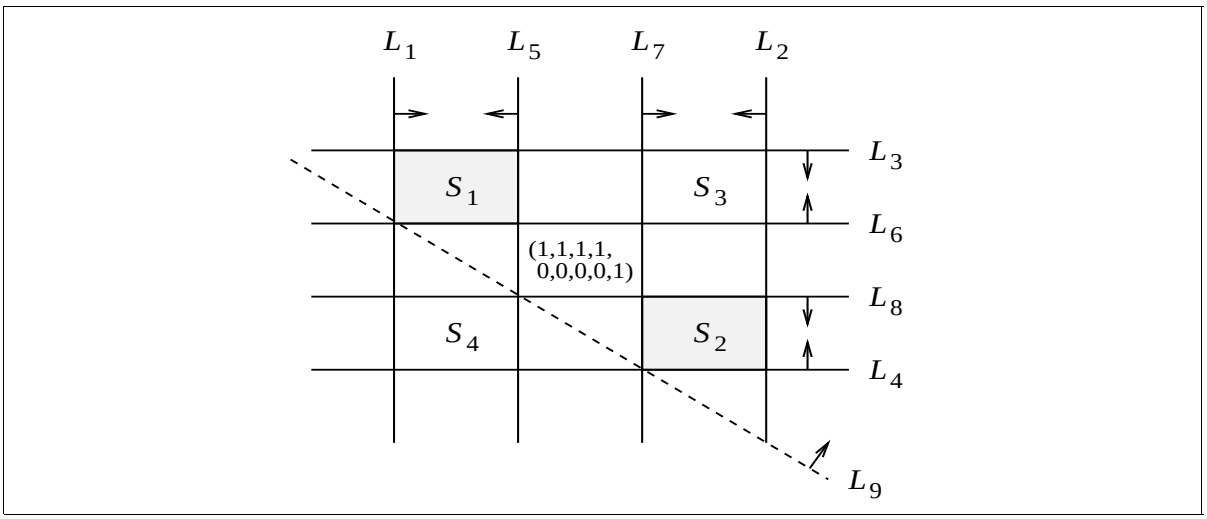

Figure 8: Counterexample to the converse of Proposition 3. Let $S = S_1 \cup S_2$. Then S cannot be realized as the decision region of a two-layer Perceptron which has only the eight (necessary) units with decision boundaries $L_1, \cdots, L_8$. However, by adding a first-layer unit with decision boundary L_9, it becomes possible to realize S as a decision region.

unit with decision boundary

$$\sum_{i=1}^{4} 3x_i + 2x_5 + x_6 + x_7 + 2x_8 + 3x_9 = 35/2$$

will result in the desired decision region.

The example above illustrates clearly one of the major difficulties of working with neural networks, namely how to determine the "correct" number of hidden units. So far we have been analyzing the capabilities of two-layer Perceptrons in terms of decision regions, decision boundaries, etc., and we found, unfortunately, that the necessary number of hidden units depends on topological properties of the function that the Perceptron is intended to realize. If we, however, instead focus on the capabilities in terms of learnability it is possible to determine a least upper bound on the necessary number of hidden units. This we shall do in Proposition 4, but first we have to define exactly what we mean by learnability and function realization.

Let f be a function defined on a finite set D in the n-dimensional Euclidean space. Let $\mathcal{P}^{(2)}(\mathbf{w})$ be a two-layer Perceptron as defined above, with an adjustable parameter (weight) vector $\mathbf{w}$. We shall say that $\mathcal{P}^{(2)}(\mathbf{w})$ is capable of realizing f if there exists a $(\mathbf{w})$ such that $\mathcal{P}^{(2)}(\mathbf{w}, \mathbf{x}) = f(\mathbf{x})$, for any $\mathbf{x} \in D$. The problem of learnability is thus to determine the number of hidden units m, such that $\mathcal{P}^{(2)}(\mathbf{w})$ is capable of realizing *arbitrary* functions f defined on a k-element set $D \subseteq \mathbb{R}^n$ where $k > 1$.

Proposition 4: Let D be a k-element set of $\mathbb{R}^n$, where $k > 1$, and let f be an arbitrary binary-valued function defined on D, i.e. $f : D \mapsto \{0, 1\}$. Then a two-layer Perceptron $\mathcal{P}^{(2)}$ with $k-1$ hidden units is capable of realizing f.
Furthermore: $m = k-1$ is the least upper bound (lub) on the number of hidden units for a $\mathcal{P}^{(2)}$ to be capable of realizing arbitrary functions defined on a k-element set $D \subseteq \mathbb{R}^n$.

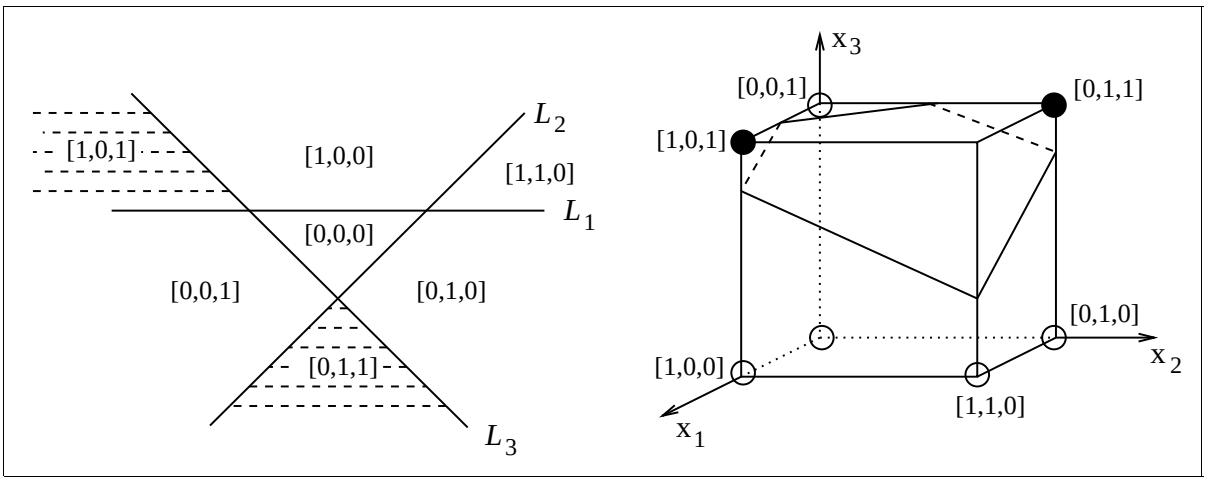

Figure 7: Geometric illustration of Proposition 2. The union of the two regions [1,0,1] and [0,1,1] can be realized as the decision region of two-layer Perceptron because the polyhedral set corresponding to [1,1,1] is empty.

Proposition 3: Let S be the decision region of $\mathcal{P}^{(2)}$ with decision boundary $B(S)$ and let $L \subseteq I\!R^n$ be a hyperplane. Suppose there exists a vector $\mathbf{x} \in B(S)$ such that $N(\mathbf{x}) \subseteq L$, where $N(\mathbf{x}) \subseteq B(S)$ is a neighborhood of $\mathbf{x}$ (in the subspace topology on $B(S)$). Then there is a first-layer unit with decision boundary L.

Put more verbally, Proposition 3 says that the number of first-layer units (or independent single-layer Perceptrons) in a two-layer Perceptron $\mathcal{P}^{(2)}$ with decision region S, has to be at least the number of hyperplanes which constitute the decision boundary $B(S)$. Indeed these units are necessary for the formation of a given decision region. Now, one might ask whether these units are also sufficient. That is, if $S \subseteq I\!R^n$ can be realized as the decision region of a two-layer Perceptron, and $L_1, \cdots, L_m$ constitute the set of all hyperplanes satisfying the conditions of Proposition 3, can S be realized, in general, as the decision region of a two-layer Perceptron with only these m first-layer units which define the decision boundaries $L_1, \cdots, L_m$? In order to illustrate that this is not the case, let us take a look at Figure 8 which provides a counterexample [8].

Consider the shaded region $S = S_1 \cup S_2$. By Proposition 3 any two-layer Perceptron which has this decision region must have eight first-layer units with decision boundaries $L_1, \cdots, L_8$ as these hyperplanes are essential for the creation of the desired boundary. It is, however, easy to see that any $\mathcal{P}^{(2)}$ which has only these eight units in its first layer cannot create a decision region which includes S_1 and S_2 but excludes S_3 and S_4. But by adding a unit with decision boundary L_9, a hyperplane which is not essential for the creation of the desired decision boundary, it becomes possible to realize S as the decision region of a two-layer Perceptron. Using the correspondence with hypercube vertices as defined by the labeling of the central region in Figure 8, it can be verified that an output

[8] Actually Gibson and Cowan proves that the converse of Proposition 3 *is* true in the special case $n = 1$. That is, if $S \subseteq I\!R$ is a finite union of disjoint, closed intervals, then S can be realized as the decision region of a two-layer Perceptron.

Focusing on the capabilities of two-layer Perceptrons in general, Proposition 1 states the necessary conditions for a region in $\mathbb{R}^n$ to be realized as a decision region for such networks. But having the geometric correspondence between polyhedral regions and hypercube vertices in mind, we can do better than that. Proposition 2 states both the necessary and the sufficient conditions for a given union of polyhedral sets defined by $L_1, L_2, \cdots, L_m$ to be realizable as the decision region of a two-layer Perceptron.

Proposition 2: Let $\mathcal{P}^{(2)}$, $L_1, \cdots, L_m$, and S be defined as described above and let $\mathbf{v}_1, \cdots, \mathbf{v}_k$ be the hypercube vertices which correspond to the polyhedral sets in the union which constitutes S. Then S can be realized as the decision region of $\mathcal{P}^{(2)}$ if and only if there exist a (possibly empty) set of hypercube vertices $\mathbf{v}_{k+1}, \cdots, \mathbf{v}_{k+r}$, such that

(**a**) $\mathbf{v}_1, \cdots, \mathbf{v}_{k+r}$ constitute a hyperplane cut of H^m, and
(**b**) the hypercube vertices $\mathbf{v}_{k+1}, \cdots, \mathbf{v}_{k+r}$ corresponds to empty intersections of the half-spaces defined by $L_1, \cdots, L_m$.

Proof: If we recall that the function which $\mathcal{P}^{(2)}$ specifies can be considered as a composite function $f = f_2 \circ f_1$, this is obvious. As we have illustrated above, the function f_2 is computed by a single-layer Perceptron and therefore its decision region is defined by a hyperplane separation in its input domain which happens to be the vertices of the hypercube H^m. That is to say that the inputs to the output unit must necessarily be vertices of H^m, and the set of vertices of H^m which, when presented as input to the output unit result in 1 being output, must constitute a hyperplane cut of H^m. □

In order to illustrate Proposition 2 geometrically, consider the regions [1,0,1] and [0,1,1] in Figure 7. Clearly these two vertices do not in themselves constitute a hyperplane cut of H^3, but the addition of the vertex [1,1,1], whose associated polyhedral set in $\mathbb{R}^2$ is empty, remedies this. Thus the union of the two regions corresponding to [1,0,1] and [0,1,1] can be realized as the decision region of a two-layer Perceptron with three hidden units[6].

Now that we have laid down the topological capabilities regarding the decision region of two-layer Perceptrons, let us direct our attention to the problems of network configuration. Similarly to Proposition 2, a natural question would now be whether it is possible to predetermine the necessary and sufficient number of units in the first layer, given a decision region which can be created by a two-layer Perceptron. As one might expect, this question must be answered in the negative. We shall, however, come up with a nontrivial lower bound (Proposition 3) and a least upper bound which, unfortunately, depends on the size of the training set applied to learn the network to realize the desired function (Proposition 4).

Proposition 3 states formally what is intuitively obvious and to keep things clear, we shall omit the proof in this presentation[7].

[6] Note that this decision region consists of two disjoint subregions.
[7] The proof can be found in [Gibson/Cowan, page 1591]

with a vertex of H^m, where the i'th coordinate of the vertex signifies which of the two half-spaces defined by L_i contains the particular region. This correspondence or coding of the polyhedral subsets is illustrated to the right in Figure 6 where we have used a 1 (0) as the i'th coordinate if the region is contained in the positive (negative) half-space defined by L_i, respectively.

The idea of considering the entire first-layer of units as a self-contained computational device f_1 can now be generalized to all the layers in any multi-layer feed-forward network. Put more formally, we may write the function f specified by a k-layer Perceptron $\mathcal{P}^{(k)}$ as a composite function

$$f = f_k \circ f_{k-1} \circ \cdots \circ f_1$$

where f_j, $j = 2, 3, \cdots, k$ are mappings exclusively between hypercube vertices, i.e. $f_j : \{0,1\}^{m_{j-1}} \mapsto \{0,1\}^{m_j}$.

2.3.2 Decision Regions of Two-Layer Perceptrons

Having these topological properties in mind, let us now face the more formal analysis of two-layer Perceptrons.

Proposition 1: The decision region S of a two-layer Perceptron $\mathcal{P}^{(2)}$ is a union of polyhedral sets.

Note, that this proposition does *not* say that S should be convex as it has been stated in [Lippmann], and [DARPA]. What it does say, however, is that the decision region is a *union* of convex sets.

Proof: For any input vector $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ to the Perceptron, the output vector $\mathbf{y} = (y_1, y_2, \cdots, y_m)$ computed by the units in the first layer, must be a vertex of the hypercube H^m, i.e. $f_1(\mathbf{x}) = \mathbf{y}$, where $\mathbf{y} \in V = \{\mathbf{v} \mid \mathbf{v} \in \{0,1\}^m\}$. For any given hypercube vertex $\mathbf{y}$ let $X(\mathbf{y}) \subseteq I\!R^n$ be the set of all inputs which result in $\mathbf{y}$ being output from the first layer. As illustrated in Figure 6, it is clear that $X(\mathbf{y})$ is a representation of an intersection (possibly empty) of half-spaces whose boundaries are the decision boundaries of the m independent Perceptrons which constitute the first layer. As the intersection of a finite number of closed sets is closed[5], $X(\mathbf{y})$ is a polyhedral set in $I\!R^n$. Similarly, let $Y(1) \subseteq \{0,1\}^m$ be the set of vertices of H^m which result in 1 being output from the single unit in the output layer of $\mathcal{P}^{(2)}$, i.e. $Y(1)$ is the decision region of this single-layer (output) Perceptron. It is now clear that the decision region S of $\mathcal{P}^{(2)}$ is given by

$$S = \bigcup_{\mathbf{y} \in Y(1)} X(\mathbf{y})$$

Or put verbally: The decision region of $\mathcal{P}^{(2)}$ is given by the union of those regions $X(\mathbf{y})$ where $\mathbf{y}$ is a hypercube vertex which, when presented as input to the second-layer unit, results in an output of 1. S is therefore a finite union of polyhedral sets. □

[5]Strictly speaking we have a problem with empty closed sets here, but to keep things simple we shall ignore this nontrivial case in this presentation. See [Gibson/Cowan] for the exact details.

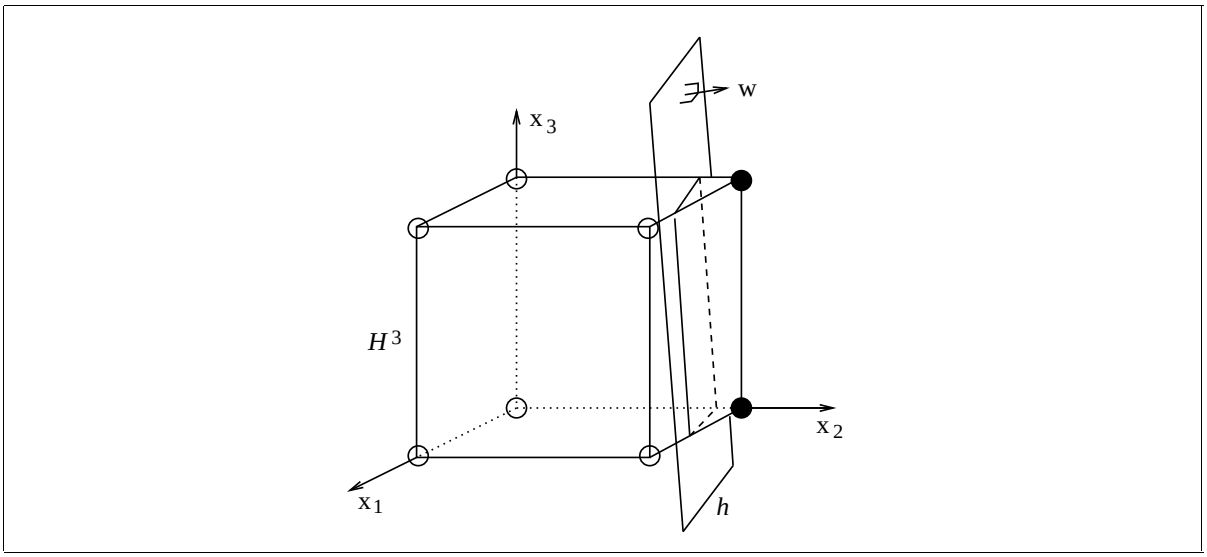

Figure 5: Geometric illustration of a hyperplane cut in $\mathbb{R}^3$. The two vertices (0,1,0) and (0,1,1) are separated from the remaining six vertices of H^3 by the hyperplane h.

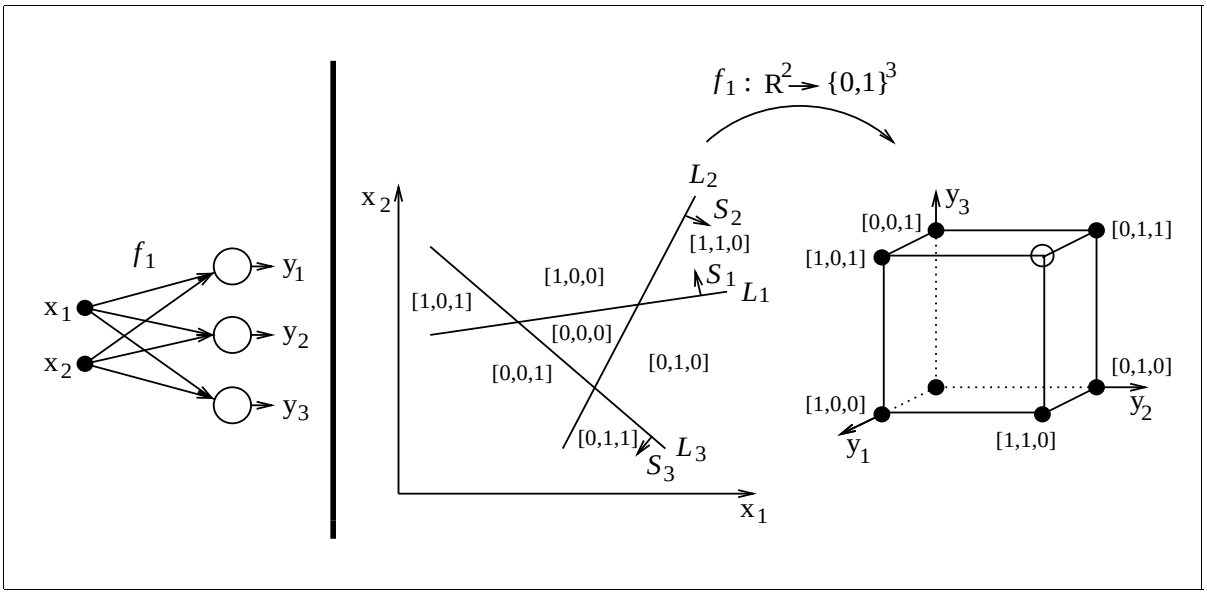

Figure 6: To the left an example of three independent single-layer Perceptrons which compute a function $f_1 : \mathbb{R}^2 \mapsto \{0,1\}^3$. To the right f_1 is illustrated geometrically using the notation defined in the text. S_i, $i = 1,2,3$, is the decision region formed by the i'th Perceptron and the three binary numbers in square brackets denote a coding of the polyhedral subsets defined by the intersections of S_1, S_2, and S_3. In the hypercube H^3 the corresponding vertices are marked with a •. Notice that $S_1 \cap S_2 \cap S_3$ (corresponding to [1,1,1]) is an empty subset of $\mathbb{R}^2$.

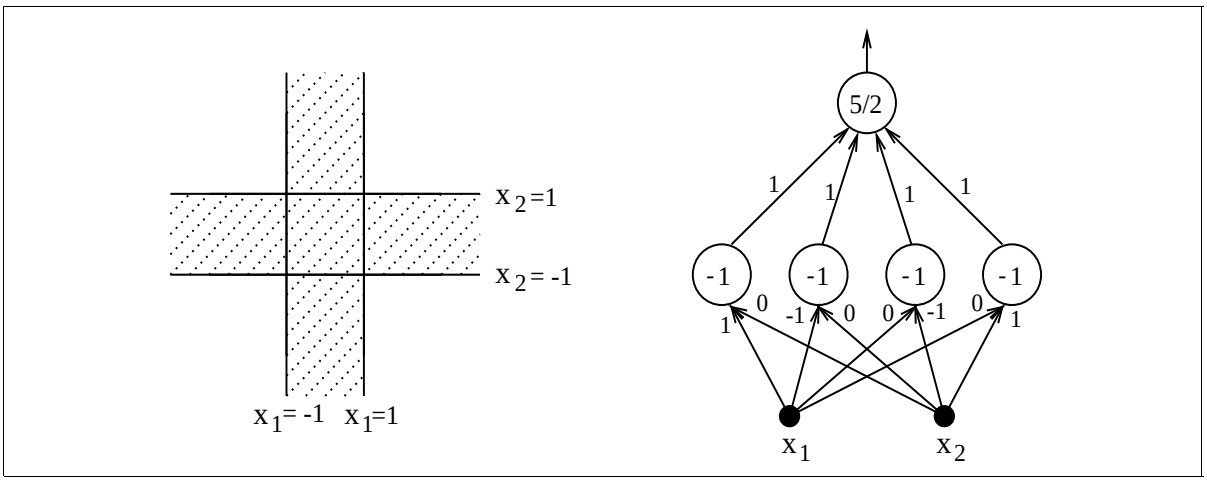

Figure 4: Two-layer Perceptron with nonconvex decision region.

2.3.1 Taxonomy and Definitions

We shall denote a two-layer Perceptron with one output unit by $\mathcal{P}^{(2)}$, and assume that it receives its inputs $x_1, \cdots, x_n$, say, through n input terminals. We restrict our attention to units with the threshold output function *sgn* defined by $sgn(x) = 1$, if $x \geq \theta$, and 0 otherwise, where θ is the "threshold" or unit bias. Such a Perceptron specifies a function $f : I\!R^n \mapsto \{0,1\}$ and, as in the preceding section, we define its *decision region* to be $S = \{\mathbf{x} \in I\!R^n \mid f(\mathbf{x}) = 1\}$. The *decision boundary* of a Perceptron with decision region S is defined to be $B(S)$, where $B(S)$ is the topological boundary of S in the usual topology of $I\!R^n$. Note, that S and $B(S)$ of a single-layer Perceptron $\mathcal{P}^{(1)}$ are, respectively, the positive half-space $\langle \mathbf{w} \cdot \mathbf{x} \rangle - \theta \geq 0$ and the hyperplane $\langle \mathbf{w} \cdot \mathbf{x} \rangle - \theta = 0$ where $\mathbf{w}$ is the weight vector.

In addition, we shall define a *polyhedral subset* of $I\!R^n$ to be a finite intersection of closed half-spaces. The m-dimensional *hypercube* H^m, is defined by

$$H^m = \{(x_1, x_2, \cdots, x_m) \in I\!R^m \mid x_i \in [0,1],\ 1 \leq i \leq m\}$$

and a *hyperplane cut* of the vertices of H^m is a subset of the vertices V, which can be separated from the remaining vertices by a single hyperplane. (See Figure 5.)

Before proceeding with the formal analysis let us illustrate how the above definitions should be geometrically interpreted. Let m denote the number of units in the first layer. Each of these m units outputs either 0 or 1, and if we for a little while consider the entire first layer as a self-contained computational device[4] which outputs the vector $\mathbf{y} = (y_1, \cdots, y_m)$, the functionality of this device, which we shall term f_1, is $f_1 : I\!R^n \mapsto \{0,1\}^m$. That is to say that the output vector $\mathbf{y}$ must be a vertex of the hypercube H^m. In Figure 6 this is illustrated for $n = 2$ and $m = 3$.

Let $L_1, L_2, \cdots, L_m$ denote the hyperplane decision boundaries of the m first-layer units. As we shall see in Proposition 1 below in a little while, any region of space expressible as an intersection of m half-spaces with boundaries $L_1, \cdots, L_m$ can be identified

[4]Actually it is m independent single-layer Perceptrons.

computes the Boolean AND function. In the light of the definitions above we may geometrically consider a single-layer Perceptron as a computational device which classifies all input vectors into one of two classes defined by the separating hyperplane L given by Equation 1. That is to say that a single-layer Perceptron with n inputs and "appropriately" adjusted weights is capable of solving any linearly separable classification problem in $\mathbb{R}^n$. This class of problems, however, is rather limited and as Minsky and Papert showed in 1969, a single-layer Perceptron is not even capable of solving the simple XOR problem. From a practical point of view one might therefore argue (and indeed nearly all researchers did in the seventies) that single-layer Perceptrons are quite uninteresting as their "computational capabilities" are absolutely inadequate for real world problems.

However, when the backpropagation learning algorithm was introduced in the mid eighties, it became possible to train feed-forward networks with hidden units, and it has since been proved [Hornik *et al.*], [Lorentz] that feed-forward networks with three layers of units are capable of approximating any continuous function arbitrarily well. And as a multi-layer feed-forward network basically can be considered as a synthesis of a number of single-layer Perceptrons, it is crucial to fully understand the functioning of the Perceptron in order to lay down the capabilities of multi-layer feed-forward networks. This is our basic approach to the study carried out in the rest of this thesis.

In the following section we shall go into details about two-layer Perceptrons and we shall analytically as well as geometrically examine the capabilities of this class of nontrivial feed-forward networks.

2.3 Capabilities of Two-Layer Perceptrons

It has been stated in the literature [Lippmann], [DARPA] that two-layer Perceptrons with one output unit are capable of forming decision regions which should be (or else be complementary to) a polyhedral set in $\mathbb{R}^n$. It has been demonstrated since, however, that this is not the whole truth [Wieland *et al.*], [Huang *et al.*], and this can be seen from the simple example in Figure 4 of a two-layer Perceptron with a decision region which is certainly not convex.

In this section we shall restrict our attention, in the main, to Perceptrons of only two layers with a single unit in the output layer (as in Figure 4). The purpose is to gain a thorough insight into the capabilities of such relatively simple networks and to demonstrate that they actually are somewhat more versatile than was originally believed. However, before we proceed we remark that the results obtained can be extended, with only minor modifications, to include two-layer Perceptrons with multiple outputs, although we do not consider such networks in this section.

Most of the analytical results in this section originate from an excellent article by Gibson and Cowan [Gibson/Cowan], but also the papers [Hao *et al.*] and [Huang/Huang] have contributed to the presentation. We have, however, tried to substantiate these results with a geometric interpretation of the basic ideas, as we believe that one should not underestimate the nice properties of the single-layer Perceptron as the basic building block of multi-layer Perceptrons.

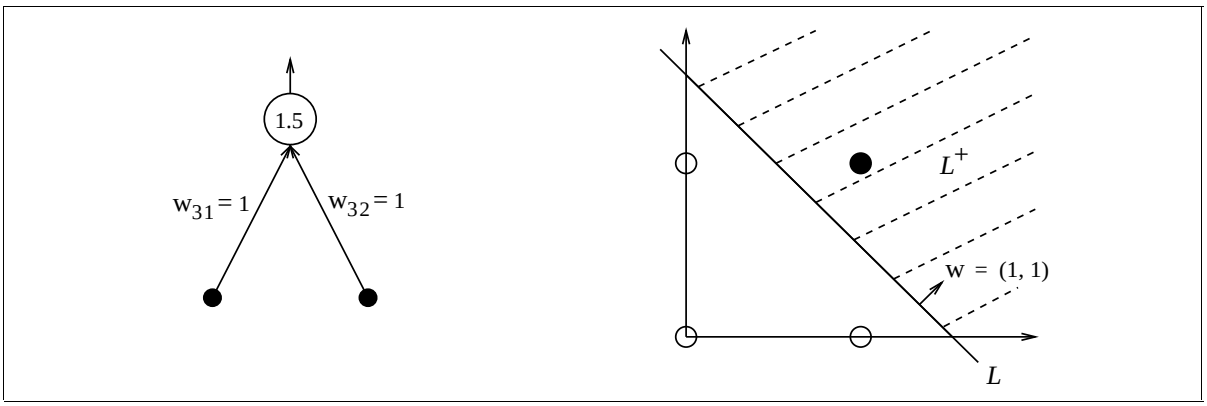

Figure 2: To the left a 2-input single-layer Perceptron for computing the Boolean function AND, and to the right the corresponding hyperplane defining the decision region L^+. The threshold value θ is 1.5, and the weight vector $\mathbf{w}$ is (1,1). A $\bullet$ ($\circ$) corresponds to TRUE (FALSE), respectively, and represent the desired output values.

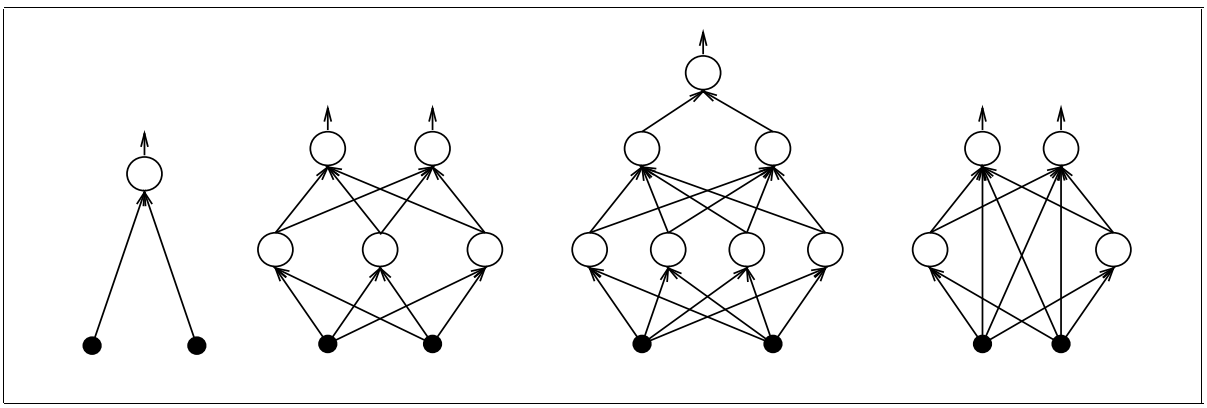

Figure 3: Examples of feed-forward networks.

interconnection between units within the same layer. The input from the environment is fed directly into the input terminals[3] which distribute the input to all the units in the successive layers to which they are connected.

The name 'feed-forward network' arises from the fact that there is a partial ordering on the connections between the units, restricting the information in the network to flow in one direction: Each layer recursively computes an output and propagates a signal to successive layers. In this sense a feed-forward network can be pictorially described as a directed acyclic graph, and Figure 3 shows some typical examples. More formally we can define a feed-forward network $\mathcal{F}$ as a special $\mathcal{N}$ in which units in layer l may only receive their inputs from units in the preceding layers $l-1, l-2, \cdots, 1$.

The simplest and most basic class of feed-forward networks is single-layer Perceptrons: A *single-layer Perceptron with one output unit* (or just *single-layer Perceptron*) [Rosenblatt], [Minsky/Papert] is a feed-forward network with n input terminals and a single output unit. Figure 2 shows a single-layer Perceptron with two inputs which

[3]We shall use the term 'input terminals' about input units as these units do not carry out any computation.

formally, as they are referred to continually in the rest of this thesis. The threshold function is defined as

$$sgn(a_i) = \begin{cases} 1 & \text{if } a_i \geq \theta \\ 0 & \text{otherwise} \end{cases}$$

where $\theta \in \mathbb{R}$ is the threshold value: only if the activation value a_i is above θ, the unit outputs 1. With n inputs the n-dimensional input weight vector for unit i

$$\mathbf{w}_i = (w_{i,1}, w_{i,2}, \cdots, w_{i,n})$$

along with the threshold value[2] θ_i defines a hyperplane L_i in $\mathbb{R}^n$, which has the normal vector $\mathbf{w}_i$

$$L_i = \{ (o_1, o_2, \cdots, o_n) \in \mathbb{R}^n \mid \sum_{j=1}^{n} w_{i,j} o_j = \theta_i \} \tag{1}$$

The corresponding positive half-space L_i^+, defined as

$$L_i^+ = \{ (o_1, o_2, \cdots, o_n) \in \mathbb{R}^n \mid \sum_{j=1}^{n} w_{i,j} o_j \geq \theta_i \} \tag{2}$$

defines what we shall call the *decision region* of unit i, and it is now clear that the threshold function sgn is 1 if and only if the input vector falls in L_i^+.

The asymmetric sigmoid is functionally similar to the threshold function, and it is defined as

$$asigm(a_i) = \frac{1}{1 + \exp^{(\theta - a_i)/T_i}}$$

where $T_i > 0$ is a "temperature" which defines the steepness of the function in an interval around the threshold value. As illustrated in Figure 1, $asigm$ approximates sgn arbitrarily well for $T_i \to 0$, and a hyperplane is still used to separate the inputs, except that the output is now continuous. The line $asigm(a_i) = 0.5$ corresponds to the same hyperplane defined by the threshold function, and in a sense the output can be thought of as a coding of the distance between the input vector and the hyperplane. A simple network for computing the Boolean function AND is illustrated in Figure 2 together with the corresponding separating hyperplane L defined by $(\mathbf{w}, \theta)$.

2.2 Feed-Forward Networks

The class of networks that are used for the simulations in this thesis are *feed-forward networks*, as presented in e.g. [PDP]. Feed-forward networks are a general category of networks that include Perceptrons and a wide class of multi-layered networks, which probably is the class of networks that has attained most research interest in the last decade. Briefly we can describe a feed-forward network as a computational device consisting of a set of interconnected units which usually are divided into layers. All units within a layer may be connected to units in all the successive layers, whereas there may be no

[2]Often θ_i is called the bias of unit i, and if we define an external unit u_0 which constantly outputs -1 (i.e. $o_0 \equiv -1$), we may consider θ_i as an extra input weight $w_{i,0}$ to unit i: $\sum_{j=1}^{n} w_{i,j} o_j - \theta_i = \sum_{j=0}^{n} w_{i,j} o_j$.

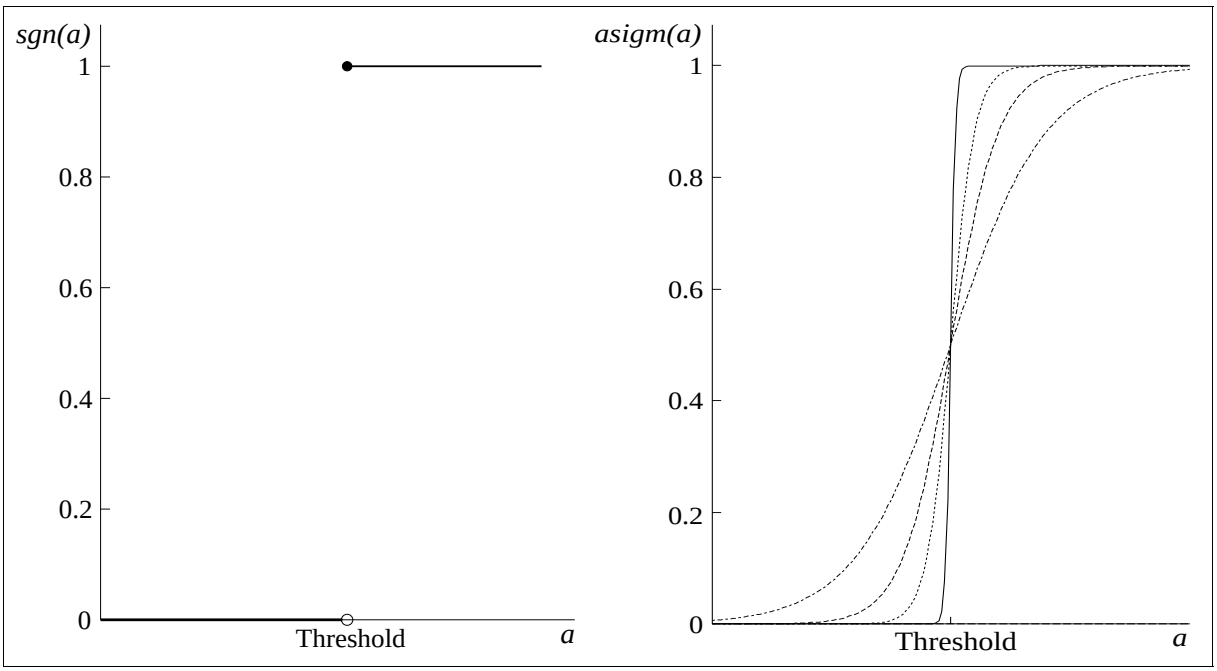

Figure 1: Two common output functions: To the left the threshold function sgn, and to the right the asymmetric sigmoid function $asigm$ for various values of the "temperature" coefficient T. When T approaches 0, $asigm$ approximates sgn arbitrarily well.

$$U \quad : \quad \{\, u_i \mid u_i \text{ is the } i\text{'th unit (or node) in the network} \,\}$$

$$W \quad : \quad \{\, w_{i,j} \mid w_{i,j} \text{ is the weight (or connection strength) to } u_i \text{ from } u_j \,\}$$

$$A \quad : \quad \{\, act_i \mid act_i \text{ is the activation function for } u_i \,\}$$

$$O \quad : \quad \{\, out_i \mid out_i \text{ is the output function for } u_i \,\}$$

This description is less "mystifying" and it contains a formal description of the distinct components that altogether constitute a network: The activation value[1] a_i for unit i represents the current state of u_i, and usually a_i is computed as the first order weighted sum of the inputs feeding into u_i

$$a_i = \sum_j w_{i,j} o_j$$

The output function out_i uses this activation value to compute the output value o_i for unit i, which then is propagated to units in the successive layers

$$o_i = out_i(a_i)$$

Figure 1 shows the graph of two commonly used output functions, namely the threshold function sgn and the asymmetric sigmoid function $asigm$. Let us define each of these

[1] We shall use a_i (o_i) to denote the value of the functions act_i (out_i), respectively.

2 Artificial Neural Networks

In recent years the field of Artificial Intelligence (AI) has seen a tremendous growth in the study of learning systems. Artificial neural networks are one class of such learning systems which have become very popular, and for the time being, artificial neural networks seem to develop into a field of its own.

This chapter concerns a general category of such networks, namely feed-forward networks. Feed-forward networks are probably the class of networks that has gone through the largest developments in the field of artificial neural networks: Until the early eighties, it was not possible to work with networks that had more than one layer of units, since no algorithm existed which could deal with more than one layer of weights. This was due to the well-known credit-assignment problem, which rendered impossible for the techniques known at that time to adjust the strength of the connections that did not lead into the output units. However, by the introduction of the generalized delta-rule, as presented in e.g. [PDP], the credit-assignment problem was overcomed and multi-layered feed-forward networks became subject of a massive research interest. Soon the category of feed-forward networks was divided into a number of subclasses, and especially the class of two-layer Perceptrons attained much research interest. The capabilities of this relatively simple class of networks have been studied in a large number of papers as e.g. [Minsky/Papert], [Hornik *et al.*], [Lippmann], [DARPA], [Wieland *et al.*], [Huang *et al.*], [Gibson/Cowan], [Hao *et al.*], and [Huang/Huang]. In this chapter we have collected the most important results which we will present from a geometric point of view. Most of the results concern the serious problems of network design, and at the end of the chapter we argue for some of the advantages of focusing on dynamic neural networks since they elegantly overcome most of these problems.

Section 2.1 concerns a general introduction to artificial neural networks, and the most basic notation that is used throughout this thesis is introduced. Section 2.2 contains a more thorough description of the general category of feed-forward networks, and, finally, Section 2.3 is devoted to the geometric presentation of the capabilities of two-layer Perceptrons.

2.1 Introduction to Artificial Neural Networks

As the title indicates, the research in artificial neural networks was originally motivated by the study of the nervous system in biological organisms (neuroscience) and by the possibility of making artificial computing models of networks of brain-like neurons. However, seen from a neurophysiological point of view, the network models considered today (and especially the basic processing element, the artificial neuron) are extremely simplified. The use of biological terms as for example "neuron," "synapse," and "axon" seems therefore to be improper in the self-contained field of artificial neural networks. We believe that such biological terms in a non-biological context is unfortunate, and instead we prefer to think of an artificial neural network $\mathcal{N}$ as a 4-tuple (U, W, A, O), where

Acknowledgements

First of all, I would like to thank my superviser, professor Brian Mayoh, for his support and valuable advice. Thanks are also due to Martin Møller and professor Miklos Szilagyi for inspiring discussions and valuable comments on a draft of the thesis.

Finally, I would like to thank Dr. Fahlman and Mr. Crowder of CMU for making the code for the Cascade-Correlation Learning Architecture available.

with are quite diverse, and therefore a direct comparison of the two algorithms might seem rather difficult. However, it will be demonstrated that the conceptual framework make possible to describe the two different network construction strategies and thereby to perform a direct comparison. Furthermore, the two algorithms are analyzed empirically on a nontrivial geometric classification problem, and it is demonstrated how the obtained results can be explained and discussed in terms of the conceptual framework.

Organization of the Thesis

Chapter 2 is a general introduction to artificial neural networks and especially to the class of feed-forward networks called two-layer Perceptrons. First, it presents the most basic notation that is used throughout the thesis. Then the capabilities of two-layer Perceptrons and the serious problems of network design are discussed thoroughly from a geometric point of view.

Chapter 3 concerns the conceptual approach to generalization. First, the basic ideas of Mitchell's work are surveyed and the corresponding conceptual framework defined. Then the chapter argues for the advantages of applying the framework to the field of artificial neural networks. It describes the adaptation of the framework to artificial neural networks in general, and afterwards it discusses the advantages of applying the framework on dynamic neural networks. Furthermore, a simple example is presented.

Chapter 4 introduces the concept of feature detectors, and the two dynamic learning algorithms are described thoroughly. Then the chapter discusses the capabilities of the two learning algorithms, and a direct comparison in terms of the conceptual framework is performed.

Chapter 5 is devoted to the empirical analyses of the two learning algorithms. Usually when a new learning algorithm is proposed, one compares its performance with results obtained by other researchers on, say, standard backpropagation. But there are good arguments against doing so (page 41), and therefore we have decided to arrange a series of experiments which are as independent as possible of results obtained by other researchers. First chapter 5 therefore describes the geometric classification problem that shall make the basis for the subsequent empirical studies, and the methodology underlying the experiments is discussed. Then the terminology that is used in the following experiments is introduced, and in order to familiarize the reader with the two learning algorithms, a typical run for each of them is presented. Finally, the two experiments we have carried out and the results obtained are presented. The first experiment concerns the scaling behavior of the two network types, while the other experiment concerns a closer analysis of the representation that the two network types utilize for found generalizations.

Chapter 6 is the conclusion. It contains a summary of the most important results, and a suggestion for a symbiosis of the two dynamic learning algorithms is presented.

There are three appendices. Appendix A contains the data underlying the results for the two typical runs. Appendix B contains all the data for Experiment 1 about the scaling behavior of the two network types, and finally, Appendix C contains all the relevant data for Experiment 2.

1 Introduction

The main objective of this thesis is to propose a conceptual approach to generalization in artificial neural networks. The conceptual framework introduced is independent of specific learning and generalization strategies, and it therefore provides a terminology which render possible to compare and discuss alternative classes of learning algorithms, and to describe the different representations for learned generalizations they employ. Furthermore, we find that the introduction of the basic idea of viewing the problem of generalization as a search problem is intuitively appealing, and it is our hope that it will provide the reader an improved comprehension of the process of generalization in artificial neural networks.

One capability central to many kinds of learning is the ability to generalize. Especially in the field of Machine Learning the problems of learning and generalization have received considerable attention. One of the most important contributions to the study of generalization in this field is probably the paper "Generalization as Search" by Tom Mitchell from 1982 [Mitchell82]. His seminal work and the idea of viewing the problem of forming a general description of a class of objects, given a set of examples and non-examples, as a search problem, have been a considerable source of inspiration to the research in Artificial Intelligence for the last decade. But also his introduction of a single conceptual framework in terms of which it is possible to compare and discuss different generalization strategies has shown its worth. The framework render possible to uncover differences and similarities among alternative approaches and it allows a precise comparison of systems that utilize different representations for learned generalizations. Especially the *version space* concept has proven to be valuable in this coherence. In this thesis it is demonstrated how the conceptual framework can be applied to the field of artificial neural networks.

It is widely known that an artificial neural network's ability to generalize depends on the configuration of the network. Since the introduction of learning algorithms that can deal with multi-layered network structures, the problems of network design have prompted much research interest, and in recent years a new class of learning algorithms which dynamically build up the networks during the training process has been introduced. This class of learning algorithms is called *dynamic learning algorithms* or *network construction algorithms*, and in this thesis the conceptual framework introduced above is used to analyze and describe such algorithms and to compare them to "traditional" learning algorithms dealing with static networks. It is shown that dynamic learning algorithms in many ways are preferable to the class of traditional learning algorithms. And in the latter half of the thesis, the framework is used to analyze and compare two specific dynamic learning algorithms and the different network construction strategies they employ. The first algorithm, *The Cascade-Correlation Learning Architecture*, is developed by Fahlman and Lebiere in 1990 [Fahlman *et al.*], while the other algorithm is developed by the author as a part of the work presented in this thesis. The two algorithms are to a large extent similar as they both employ exactly the same learning procedure to train the networks. They differ, however, decisively in the way they build up these networks during the training process: The Cascade-Correlation Learning Architecture builds multi-layer feed-forward networks with as many layers as the number of added hidden units, while the other algorithm operates with only one layer of hidden units. This implies that their computational capabilities and the representation of the learned generalizations they deal

Contents

1 Introduction **1**

2 Artificial Neural Networks **4**
2.1 Introduction to Artificial Neural Networks 4
2.2 Feed-Forward Networks 6
2.3 Capabilities of Two-Layer Perceptrons 8
2.3.1 Taxonomy and Definitions 9
2.3.2 Decision Regions of Two-Layer Perceptrons 11

3 The Process of Generalization **18**
3.1 A Conceptual Approach to Generalization 18
3.2 Generalization in Static Neural Networks 21
3.2.1 Biases 24
3.3 Generalization in Dynamic Neural Networks 26
3.3.1 Example: The Boolean XOR Function 27

4 Two Dynamic Learning Algorithms **30**
4.1 Feature Discovery 30
4.2 Description of Two Cascade-Correlation Learning Algorithms 31
4.2.1 The High-Order Cascor Algorithm 32
4.2.2 The Second-Order Cascor Algorithm 35
4.3 The Two Cascade-Correlation Learning Algorithms Revisited 35
4.3.1 Analysis of the Two Distinct Hypothesis Space Construction Techniques 36

5 Empirical Analyses **41**
5.1 Description of the Three-Disc Problem 41
5.2 Methodology 44
5.2.1 Initial Weights 44
5.2.2 Stopping Criterion and Learning Parameters 44
5.2.3 Criteria of Correct Classification 46
5.2.4 Training and Test Sets 47
5.3 A Typical Run 48
5.4 Experiment 1: Scaling Behavior 54
5.5 Experiment 2: Improving the Learnability 60

6 Conclusion **67**

A Appendix **75**

B Appendix **78**

C Appendix **81**

A Conceptual Approach to Generalization in Dynamic Neural Networks

By

Steen Sjøgaard

Computer Science Department, Aarhus University
DK-8000 Aarhus C, Denmark
E-mail: steensj@daimi.aau.dk

October 1991