Chapter 9

Backpropagation on the MONK's Problems

Sebastian B. Thrun

Carnegie-Mellon University, School of Computer Science, Pittsburgh, PA 15213
e-mail: Sebastian.Thrun@cs.cmu.edu

9.1 Introduction

This paper briefly describes the results of the plain backpropagation algorithm [1] obtained on the three MONK's problems. Backpropagation is a function approximation algorithm for multilayer feed-forward perceptrons based on gradient descent. Conversely to many symbolic learning algorithms, backpropagation learns functions by nonlinear L_2-approximations. This technique has been successfully applied to a variety of real-world problems like speech recognition, bomb detection, stock market prediction etc.

Although multilayer networks represent continuous functions, they are frequently restricted to binary classification tasks as the MONK's problems. In all three cases we used the following architecture: There were 17 input units, all having either value 0 or 1 corresponding to which attribute-value was set. All input units had a connection to 3 (first MONK's problem), 2 (second problem) or 4 (third problem) hidden units, which itself were fully connected to the output unit. An input was classified as class member if the output, which is naturally restricted to $(0, 1)$, was $\geq .5$. Training took between ten and thirty seconds on a SUN Sparc Station for each of the three problems. On a parallel computer, namely the Connection Machine CM-2, training time was further reduced to less than 5 seconds for each problem. The following results are obtained by the plain, unmodified backpropagation algorithm. These results reflect what an unexperienced user would obtain by running backpropagation on the MONK's problems.

	training epochs	accuracy
MONK's # 1	390	100%
MONK's # 2	90	100%
MONK's # 3	190	93.1%

However, in the third training set, the error did never approach zero in all runs we performed, which indicated the presence of noise and/or a local minimum. This important observation led us to refine the results for the third problem using *weight decay*[1] [1,2]. This wideley used technique often prevents backpropagation nets from overfitting the training data and thus improves the generalization. With weight decay $\alpha = 0.01$ we improved the classification accuracy on this third set significantly and, moreover, the concept learned was the same for all architectures we tested (i,e, 2, 3, or 4 hidden units).

	training epochs	accuracy
MONK's # 3 with weight decay	105	97.2%

Backpropagation with weight decay learned the correct concepts for the first two MONK's problems again with 100% accuracy. These classification results clearly demonstrate the appropriatenes of the backpropagation algorithm on problems as the MONK's problems.

References

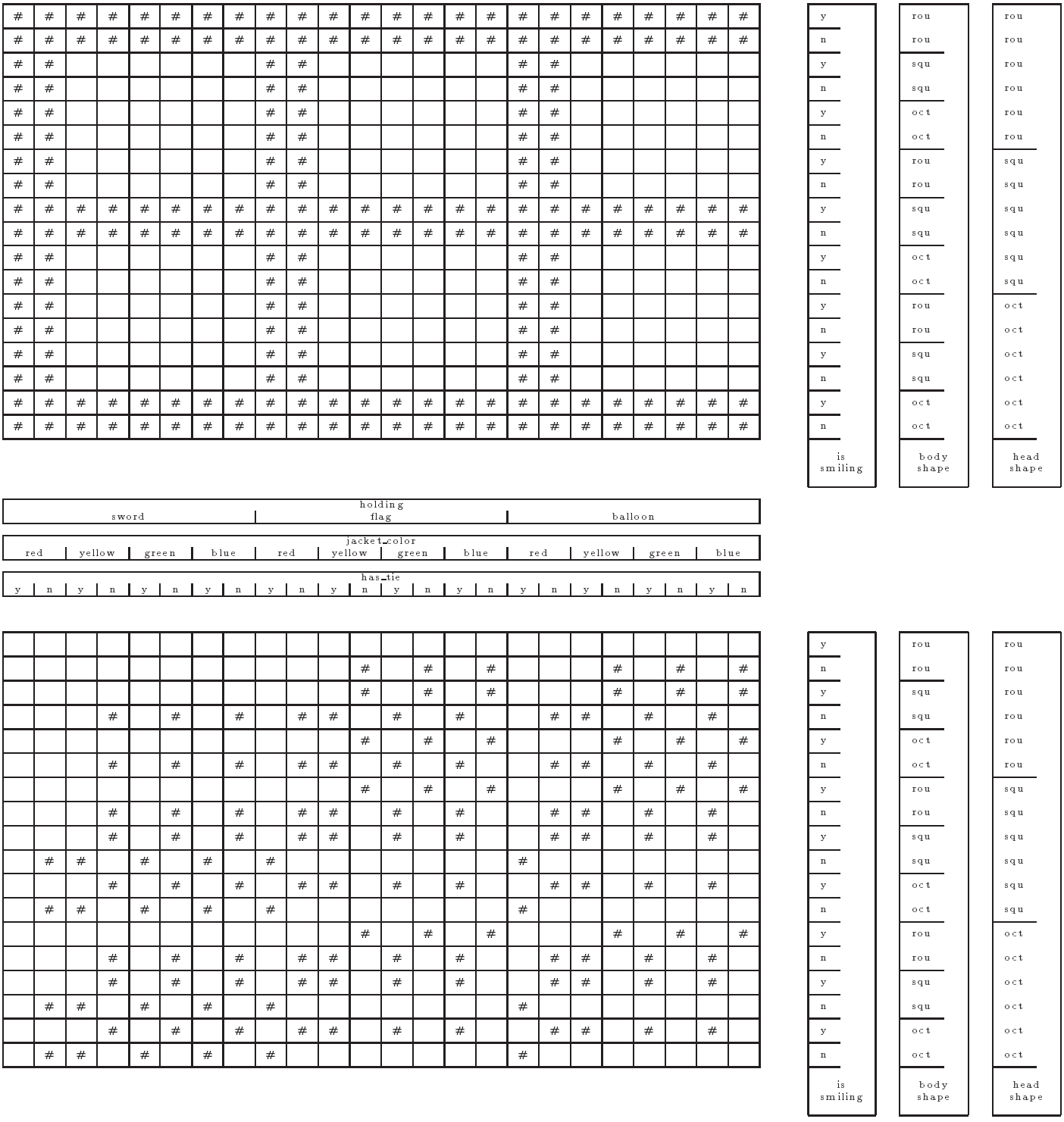
[1] Rumelhart, D. E. and McClelland, J. *Parallel Distributed Processing. Vol. I + II*, MIT Press 1986

[2] Chauvin, Y. Dynamic Behavior of Contrained Backpropagation Networks. In *Advances in Neural Information Processing Systems 2.* Morgan Kaufmann Publishers, 1990.

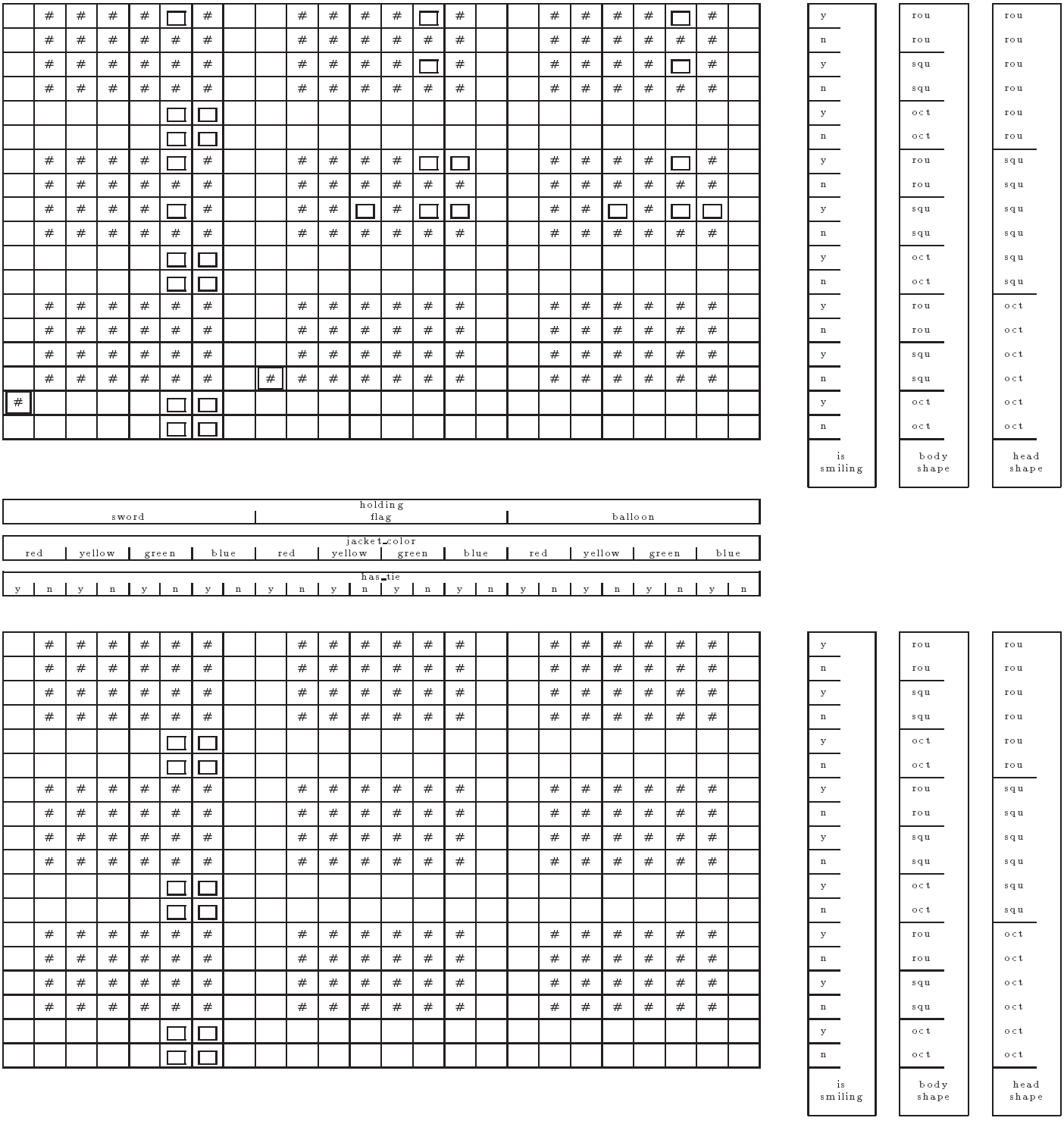
[1] In our implementation, weight decay was realized by minimizing the complexity term $\alpha \cdot \frac{1}{4}(\sum_{i,j} w_{ij}^4 + \sum_i \theta_i^4)$ in addition to the conventional L_2-error term over the training set. Here α is a constant factor, w_{ij} denotes the weight from unit j to unit i, and θ_i the threshold (bias) of unit i.

9.2 Classification diagrams

(a) Results of BACKPROP (with/without weight decay) on test set 1, Accuracy: 100.0%
(b) Results of BACKPROP (with/without weight decay) on test set 2, Accuracy: 100.0%



(a) Results of BACKPROP without weight decay on test set 3, Accuracy: 93.1%
(b) Results of BACKPROP with weight decay on test set 3, Accuracy: 97.2%



9.3 Resulting weight matrices

MONKS's problem # 1: weights and biases

	to-node			
from-node	hidden_1	hidden_2	hidden_3	output
input_1 (head_shape round)	-6.503145	0.618412	-1.660409	
input_2 (head_shape square)	1.210703	1.939613	2.972592	
input_3 (head_shape octagon)	5.356444	-3.597301	-1.266992	
input_4 (body_shape round)	-6.692434	2.129635	-2.032242	
input_5 (body_shape square)	6.457639	0.864312	4.260765	
input_6 (body_shape octagon)	0.225053	-2.428098	-1.839603	
input_7 (is_smiling yes)	0.096995	0.131133	0.053480	
input_8 (is_smiling no)	-0.011828	0.135277	0.107302	
input_9 (holding sword)	-0.076848	0.459903	-0.008368	
input_10 (holding balloon)	-0.016940	0.151738	0.148955	
input_11 (holding flag)	-0.087298	0.196521	0.023554	
input_12 (jacket_color red)	5.735210	4.337359	-0.865479	
input_13 (jacket_color yellow)	-2.257168	-1.410376	0.494681	
input_14 (jacket_color green)	-2.232257	-1.109825	0.382717	
input_15 (jacket_color blue)	-1.710642	-1.452455	0.479513	
input_16 (has_tie yes)	-0.109696	0.434166	0.276487	
input_17 (has_tie no)	-0.111667	0.131797	0.310714	
bias	0.486541	0.142383	0.525371	
hidden_1				9.249339
hidden_2				8.639715
hidden_3				-9.419991
bias				-3.670920

MONKS's problem # 2: weights and biases

	to-node		
from-node	hidden_1	hidden_2	output
input_1 (head_shape round)	-4.230213	3.637149	
input_2 (head_shape square)	1.400753	-2.577242	
input_3 (head_shape octagon)	1.479862	-2.492254	
input_4 (body_shape round)	-4.363966	3.835199	
input_5 (body_shape square)	1.154510	-2.347489	
input_6 (body_shape octagon)	1.542958	-2.227530	
input_7 (is_smiling yes)	-3.396133	2.984736	
input_8 (is_smiling no)	1.868955	-2.994535	
input_9 (holding sword)	-4.041057	4.239548	
input_10 (holding balloon)	1.293933	-2.195403	
input_11 (holding flag)	1.160514	-2.272035	
input_12 (jacket_color red)	-4.462360	4.451742	
input_13 (jacket_color yellow)	0.749287	-1.869545	
input_14 (jacket_color green)	0.640353	-1.727654	
input_15 (jacket_color blue)	1.116349	-1.332642	
input_16 (has_tie yes)	-3.773187	3.290757	
input_17 (has_tie no)	1.786105	-3.296139	
bias	-1.075762	-0.274980	
hidden_1			-11.038625
hidden_2			-9.448544
bias			5.031395

MONKS's problem # 3: weights and biases (without weight decay)

	to-node				
from-node	hidden_1	hidden_2	hidden_3	hidden_4	output
input_1 (head_shape round)	0.277334	-0.673423	-0.345908	0.121908	
input_2 (head_shape square)	1.759524	1.150119	0.098689	0.329486	
input_3 (head_shape octagon)	-1.328410	0.941278	0.059910	0.017674	
input_4 (body_shape round)	-3.466870	-0.022787	0.222484	0.214138	
input_5 (body_shape square)	-2.460525	3.988668	-0.021681	0.235819	
input_6 (body_shape octagon)	6.622062	-3.396938	0.125944	-0.134328	
input_7 (is_smiling yes)	1.615026	0.224221	-0.317908	-0.594920	
input_8 (is_smiling no)	-1.433791	-0.183452	-0.326539	0.361663	
input_9 (holding sword)	-0.780008	-0.786854	0.072768	0.507106	
input_10 (holding balloon)	0.733984	-0.260836	0.004670	0.422573	
input_11 (holding flag)	0.415208	1.410443	-0.023262	0.325766	
input_12 (jacket_color red)	-3.263737	1.324415	0.025837	-0.154449	
input_13 (jacket_color yellow)	-1.896538	1.518800	0.351912	0.044775	
input_14 (jacket_color green)	-0.432256	-0.183302	0.057546	-0.058255	
input_15 (jacket_color blue)	5.090627	-3.446529	-0.082472	0.131738	
input_16 (has_tie yes)	0.897500	-0.717589	0.314088	0.099872	
input_17 (has_tie no)	-0.502348	0.954327	-0.074583	-0.339295	
bias	0.364889	0.248641	-0.484047	-0.227007	
hidden_1					-11.548968
hidden_2					6.567443
hidden_3					-0.117112
hidden_4					-0.064650
bias					0.191083

MONKS's problem # 3: weights and biases (with weight decay)

	to-node		
from-node	hidden_1	hidden_2	output
input_1 (head_shape round)	-0.029477	-0.008986	
input_2 (head_shape square)	-0.376094	-0.364778	
input_3 (head_shape octagon)	-0.051924	-0.028672	
input_4 (body_shape round)	0.991798	0.991750	
input_5 (body_shape square)	1.031170	1.027708	
input_6 (body_shape octagon)	-1.284263	-1.279808	
input_7 (is_smiling yes)	-0.303940	-0.314212	
input_8 (is_smiling no)	-0.216766	-0.221040	
input_9 (holding sword)	-0.064305	-0.052110	
input_10 (holding balloon)	-0.257165	-0.243988	
input_11 (holding flag)	-0.131509	-0.122790	
input_12 (jacket_color red)	1.001415	1.004192	
input_13 (jacket_color yellow)	0.898066	0.896869	
input_14 (jacket_color green)	0.670929	0.673218	
input_15 (jacket_color blue)	-1.280272	-1.272798	
input_16 (has_tie yes)	-0.354472	-0.355268	
input_17 (has_tie no)	0.040973	0.037927	
bias	-0.319686	-0.343492	
hidden_1			1.762523
hidden_2			1.759077
bias			-1.501492

Chapter 10

The Cascade-Correlation Learning Algorithm on the MONK's Problems

Scott E. Fahlman

Carnegie Mellon University, School of Computer Science, Pittsburgh, PA 15213
e-mail: Scott.Fahlman@cs.cmu.edu

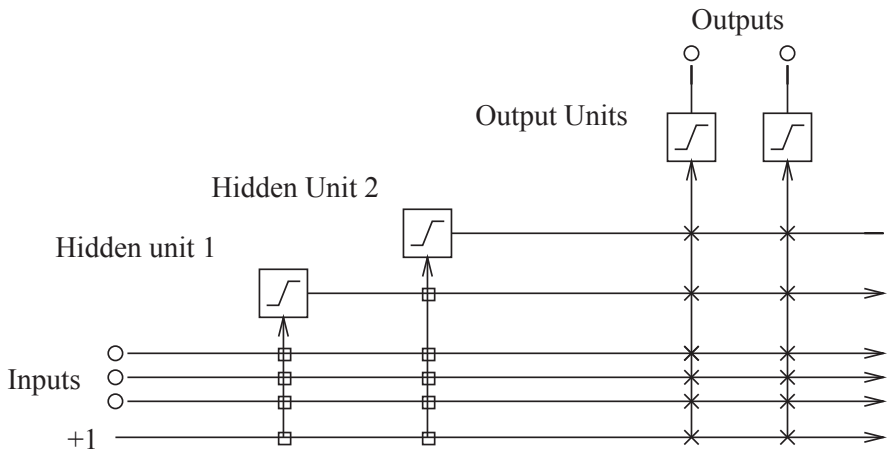


Figure 10.1: see text for details

10.1 The Cascade-Correlation algorithm

Cascade-Correlation [?] is a supervised neural network learning architecture that builds a near-minimal multi-layer network topology in the course of training. Initially the network contains only inputs, output units, and the connections between them. This single layer of connections is trained (using the Quickprop algorithm [?]) to minimize the error. When no further improvement is seen in the level of error, the network's performance is evaluated. If the error is small enough, we stop. Otherwise we add a new hidden unit to the network in an attempt to reduce the residual error.

To create a new hidden unit, we begin with a pool of *candidate units*, each of which receives weighted connections from the network's inputs and from any hidden units already present in the net. The outputs of these candidate units are not yet connected into the active network. Multiple passes through the training set are run, and each candidate unit adjusts its incoming weights to maximize the correlation between its output and the residual error in the active net. When the correlation scores stop improving, we choose the best candidate, freeze its incoming weights, and add it to the network. This process is called "tenure." After tenure, a unit becomes a permanent new feature detector in the net. We then re-train all the weights going to the output units, including those from the new hidden unit. This process of adding a new hidden unit and re-training the output layer is repeated until the error is negligible or we give up. Since the new hidden unit receives connections from the old ones, each hidden unit effectively adds a new layer to the net. (See figure 1.)

Cascade-correlation eliminates the need for the user to guess in advance the network's size, depth, and topology. A reasonably small (though not minimal) network is built automatically. Because a hidden-unit feature detector, once built, is never altered or cannibalized, the network can be trained incrementally. A large data set can be broken up into smaller "lessons," and feature-building will be cumulative.

Cascade-Correlation learns much faster than backprop for several reasons: First only a single layer of weights is being trained at any given time. There is never any need to propagate error information backwards through the connections, and we avoid the dramatic slowdown that is typical when training backprop nets with many layers. Second, this is a "greedy" algorithm: each new unit grabs as much of the remaining error as it can. In a standard backprop net, the all the hidden units are changing at once, competing for the various jobs that must be done—a slow and sometimes unreliable process.

10.2 Results

For all these problems I used the standard Common Lisp implementation of Cascade-Correlation on a Decstation 3100. This code is public-domain and is available to outside users via anonymous FTP. Contact `sef@cs.cmu.edu` for details.

I used the same parameters in all of these tests. Here is the printout of those parameters:

SigOff 0.10	WtRng 1.00	WtMul 1.00		
OMu 2.00	OEps 1.00	ODcy 0.0000	OPat 20	OChange 0.010
IMu 2.00	IEps 1.00	IDcy 0.0000	IPat 15	IChange 0.030
Utype :GAUSSIAN	Otype :SIGMOID	RawErr NIL	Pool 8	
(train 100 100 10)				

Monk #1:

After 95 epochs, 1 hidden unit: 0 Errors on training set. 0 Errors on test set.
Elapsed real time: 5.11 seconds

Monk #2:

After 82 epochs, 1 hidden unit: 0 Errors on training set. 0 Errors on test set.
Elapsed real time: 7.75 seconds

Monk #3:

After 259 epochs, 3 hidden units: 0 Errors on training set. 40 errors on test set (i.e. accuracy 95.4%).
Elapsed real time 12.27 seconds.

Training and test-set performance was tested after each output-training phase. The minimum test-set error was observed after the initial output-training phase, before any hidden units were added. (Not surprising, since with no noise this problem is linearly separable.) Using any sort of cross-validation system, this is where the algorithm would stop.

At that point, the results were as follows:

Training: 7 of 122 wrong:

Head: RND	Body: RND	Smile: Y	Holding: SWD	Jacket: GRN	Tie: Y	Output: T
Head: RND	Body: SQR	Smile: Y	Holding: BAL	Jacket: GRN	Tie: Y	Output: T
Head: SQR	Body: SQR	Smile: Y	Holding: BAL	Jacket: YEL	Tie: Y	Output: T
Head: SQR	Body: SQR	Smile: Y	Holding: FLG	Jacket: GRN	Tie: N	Output: T
Head: SQR	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: OCT	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL
Head: OCT	Body: OCT	Smile: Y	Holding: SWD	Jacket: BLU	Tie: Y	Output: NIL

Test: 14 of 432 wrong:

Head: RND	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: RND	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL
Head: RND	Body: OCT	Smile: N	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: RND	Body: OCT	Smile: N	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL
Head: SQR	Body: SQR	Smile: Y	Holding: BAL	Jacket: GRN	Tie: Y	Output: NIL
Head: SQR	Body: SQR	Smile: Y	Holding: FLG	Jacket: GRN	Tie: Y	Output: NIL
Head: SQR	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: SQR	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL
Head: SQR	Body: OCT	Smile: N	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: SQR	Body: OCT	Smile: N	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL
Head: OCT	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: OCT	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL
Head: OCT	Body: OCT	Smile: N	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: OCT	Body: OCT	Smile: N	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL

So on the test set, performance is 96.7%

By turning up the OUTPUT-DECAY parameter to 0.1 (an odd thing to do, but sometimes useful when the training set is too small for good generalization), we can do a little better. After the initial output-training phase:

Training: 8 of 122 wrong:

Head: RND	Body: RND	Smile: Y	Holding: SWD	Jacket: GRN	Tie: Y	Output: T
Head: RND	Body: SQR	Smile: Y	Holding: BAL	Jacket: GRN	Tie: Y	Output: T
Head: SQR	Body: SQR	Smile: Y	Holding: BAL	Jacket: YEL	Tie: Y	Output: T
Head: SQR	Body: SQR	Smile: Y	Holding: FLG	Jacket: GRN	Tie: Y	Output: T
Head: SQR	Body: SQR	Smile: Y	Holding: FLG	Jacket: GRN	Tie: N	Output: T
Head: SQR	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: OCT	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL
Head: OCT	Body: OCT	Smile: Y	Holding: SWD	Jacket: BLU	Tie: Y	Output: NIL

Test: 12 of 432 wrong:

Head: RND	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: RND	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL
Head: RND	Body: OCT	Smile: N	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: RND	Body: OCT	Smile: N	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL
Head: SQR	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: SQR	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL
Head: SQR	Body: OCT	Smile: N	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: SQR	Body: OCT	Smile: N	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL
Head: OCT	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: OCT	Body: OCT	Smile: Y	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL
Head: OCT	Body: OCT	Smile: N	Holding: SWD	Jacket: GRN	Tie: Y	Output: NIL
Head: OCT	Body: OCT	Smile: N	Holding: SWD	Jacket: GRN	Tie: N	Output: NIL

Score on test set: 97.2%

We can see here what the problem is: All the bad test-set cases are Green and holding a sword, so they should be true. But this positive value is not strong enough to offset the negative weight from Octagonal body.

In the training set, there are only two examples showing the green-sword combination overpowering an octagonal body, and that is apparently not enough to make the point. There are 11 cases showing that octagonal/sword should be negative and 8 cases showing that octagonal/green should be negative.

If we switch the training and test set, we see how easy it is to solve this problem in the absence of noise and

small-sample fluctuations.

Switching the training and test set: After 16 epochs and 0 hidden units:

Training: 0 of 432 wrong. Test: 6 of 122 wrong.

Head: RND	Body: RND	Smile: Y	Holding: SWD	Jacket: GRN	Tie: Y	Output: T
Head: RND	Body: SQR	Smile: Y	Holding: BAL	Jacket: GRN	Tie: Y	Output: T
Head: SQR	Body: SQR	Smile: Y	Holding: BAL	Jacket: YEL	Tie: Y	Output: T
Head: SQR	Body: SQR	Smile: Y	Holding: FLG	Jacket: GRN	Tie: Y	Output: T
Head: SQR	Body: SQR	Smile: Y	Holding: FLG	Jacket: GRN	Tie: N	Output: T
Head: OCT	Body: OCT	Smile: Y	Holding: SWD	Jacket: BLU	Tie: Y	Output: NIL

These, I believe, are exactly the noise cases deliberately inserted in the original training set. Note that three of these noise cases are

Square/Square/Yes $\Longrightarrow$ NIL (when T is correct)

This explains the other two error cases observed in the first run of this problem. If we look at square/square/yes cases in the training set, NIL cases outnumber T cases, 5 to 3.

Bibliography

[Fahlman, 1988] Fahlman, S. E. (1988) "Faster-Learning Variations on Back-Propagation: An Empirical Study" in *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann.

[Fahlman, 1990] Fahlman, S. E. and C. Lebiere (1988) "The Cascade-Correlation Learning Architecture" in D. S. Touretzky (ed.), *Advances in Neural Information Processing Systems 2*, Morgan Kaufmann.

10.3 Classification diagrams

(a) Training set #3 first run, Accuracy: 96.8%
(b) Training set #3 second run, Accuracy: 97.2%