In Proc. 1994 IEEE Workshop on Neural Networks for Signal Processing IV pp. 177-186

CLASSIFICATION USING HIERARCHICAL MIXTURES OF EXPERTS

S.R.Waterhouse A.J.Robinson
Cambridge University Engineering Department
Trumpington Street, Cambridge CB2 1PZ, England

Abstract–There has recently been widespread interest in the use of multiple models for classification and regression in the statistics and neural networks communities. The Hierarchical Mixture of Experts (HME) [1] has been successful in a number of regression problems, yielding significantly faster training through the use of the Expectation Maximisation algorithm. In this paper we extend the HME to classification and results are reported for three common classification benchmark tests: Exclusive-Or, N-input Parity and Two Spirals.

INTRODUCTION

Traditional Neural Network architectures such as the multi-layer perceptron have proved successful as universal function approximators and have been used in many different problems ranging from pattern classification to control engineering. Whilst there is undoubtably further valuable work to be done on such architectures, such as improving training methods, there is a considerable incentive to look in other directions for new architectures. Such architectures ideally would be statistically motivated and have parameters which are easily interpretable; they would also allow training speeds to be increased, since the gradient descent algorithm used in traditional back-propagation is typically too slow for solving real-world problems in real time.

Motivated by such concerns, a number of researchers have investigated methods of function approximation incorporating ideas from the fields of statistics and neural networks. One recurring trend in such work is the use of separate models to approximate different parts of a problem. The general approach is to divide the problem into a series of sub-problems and assign a set of function approximators or 'experts' to each sub-problem. Different approaches use different techniques to divide the problem into sub-problems and to calculate the best solution to the problem from the outputs of the experts. The architecture described in this paper, the Hierarchical Mixtures of Experts (HME) [1], employs probabilistic methods in both the way it divides the input space and the way it combines the outputs from the experts.

The paper is organised as follows. The HME architecture is described, along with the use of the Expectation Maximisation (EM) algorithm [2] which is used to estimate its parameters. The extension of the HME to classification is discussed, including the required modifications to the training algorithm. The results obtained on two classification simulations are presented: N-input Parity and the 'Two Spirals' problem.

HIERARCHICAL MIXTURES OF EXPERTS

The HME is based on the principle of 'divide and conquer' in which a large, hard to solve problem is divided into many smaller, easier to solve problems. There are

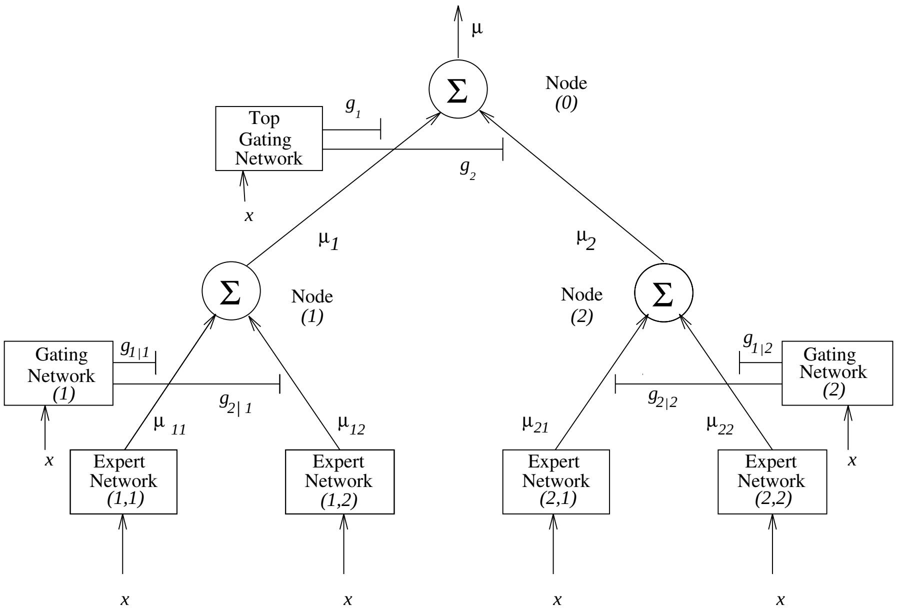


Figure 1: Hierarchical Mixture of Experts

several alternative strategies for tackling such problems. The simplest approach is to divide the problem into sub-problems having no common elements - a ‘hard split’ of the data. The optimum output of the experts assigned to each sub-problem may then be chosen on a ‘winner-takes-all’ (WTA) basis. Classification and Regression Trees (CART) [3] are based on this principle. Alternatively, the outputs of the experts may be combined in a weighted sum with weights derived from the performance of the experts in their partition of the input space; this is the principle behind Stacked Generalisation [4]. The most advanced method is to divide the problem into sub-problems which can have common elements - a ‘soft split’ of the input space into a series of overlapping clusters. The outputs can be chosen either using WTA or stochastically. The HME combines the ideas of soft splits of the data with stochastic selection of the outputs of the experts by the use of a gating network. A two-level HME architecture with a common branching factor of two at each level (a ‘binary branching’ HME) is shown in Figure 1. In the general architecture, multiple levels and branching factors are possible. In its basic form, the HME employs a fixed architecture which is pre-determined before training commences. The tree consists of non-terminal and terminal nodes which we denote by a set of indicator variables $\{Z\}$. Non-terminal node (1) is thus denoted z_1 and consists of gating network GN(1). Terminal node $(1,1)$ is denoted by z_{11} and consists of expert network EN(1,1). A general HME with i levels of gating networks and branching factors $b_0, b_1, ..., b_{i-1}$ is denoted by HME(i:$b_0, b_1, ..., b_{i-1}$), thus Figure 1 is HME$(2{:}2,2)$. In the original HME each expert was linear and performed a regression task [1]. In this paper, each expert is non-linear and performs multi-way classification.

Our general classification problem may be considered as follows. At time t during training, we observe an input vector $\boldsymbol{x}^{(t)}$ which belongs to class n. We construct a *target* output vector $\boldsymbol{y}^{(t)}$ with 1 in element n and 0 elsewhere. We wish to compute the probability $P(y_n|\boldsymbol{x}^{(t)})$ of the correct class n being returned given the input vector at time t. [1] We do this by breaking the problem into a series of smaller problems. For example, expert network EN$(1,1)$ computes $P(\boldsymbol{y}|\boldsymbol{x},z_1,z_{11})$, the probability vector of all classes given that we took the left branch of every split and ended up in terminal node $(1,1)$. The top level gating network GN(0) computes $P(z_1|\boldsymbol{x})$, and the second level gating network GN(1) computes $P(z_{11}|\boldsymbol{x},z_1)$. GN(1) weights EN$(1,1)$ and EN$(1,2)$ to give the output of node (1),

$$\mu_1 = P(\boldsymbol{y}|\boldsymbol{x},z_1) = \sum_{j=1}^{b_1} P(\boldsymbol{y}|\boldsymbol{x},z_1,z_{1j})P(z_{1j}|\boldsymbol{x},z_1).$$

All these nodes are combined to give the overall output μ of the HME,

$$\begin{aligned} \mu = P(\boldsymbol{y}|\boldsymbol{x}) &= \sum_{i=1}^{b_0} P(z_i|\boldsymbol{x})P(\boldsymbol{y}|\boldsymbol{x},z_i) \\ &= \sum_{i=1}^{b_0} P(z_i|\boldsymbol{x}) \sum_{j=1}^{b_1} P(z_{ij}|\boldsymbol{x},z_i)P(\boldsymbol{y}|\boldsymbol{x},z_i,z_{ij}) \end{aligned}$$

This process may be extended to any depth and may use arbitrary branching factors at each depth. Unlike CART, the shape of the HME network is pre-determined heuristically before training.

Expert network EN$(1,1)$ is a single layer network with ‘softmax‘ activation function [5] whose j^{th} output is

$$\mu_{11j} = P(\boldsymbol{y}|\boldsymbol{x},z_1,z_{11},\Theta_{11}) = \exp(\theta_{11j}^T\boldsymbol{x}) \Bigg/ \sum_{k=1}^{N} \exp(\theta_{11k}^T\boldsymbol{x})\ ,$$

where Θ_{11} is a parameter matrix, consisting of N independent vectors $\{\theta_{11k}\}$. The form of GN(1) is

$$P(z_1|\boldsymbol{x},\Xi_1) = \exp(\xi_{11}^T\boldsymbol{x}) \Bigg/ \sum_{i=1}^{b_1} \exp(\xi_{1i}^T\boldsymbol{x})$$

where Ξ_1 is the parameter matrix for this gate, consisting of b_1 independent vectors ξ_{1i}. Therefore, the mathematical form of the gating and expert networks is the same, with the difference that the gating network is classifying the experts over the input space and the experts are classifying within the input space regions.

Training the HME

The HME is trained using the Expectation Maximisation (EM) algorithm, in which ‘missing data’ is specified, which if known would simplify the maximisation problem. If we had information about which node had generated the data, we could update the parameters for the gates and experts for that node. Thus the missing data for the EM algorithm applied to HMEs is the set of *indicator* variables $\{Z\}$ which indicate which node generated each output, or alternatively which node is best suited to the portion

---

[1] For simplicity of notation, we shall now drop the superscript t on the inputs, outputs and indicator variables.

of the input space under consideration. The E-step of the EM algorithm reduces to computing the expected values of the indicator variables which gives the set of *posterior* probabilities $\{H\}$. The *conditional* posterior probability of node $(1,1)$ is the probability that EN$(1,1)$ can be considered to have generated the data based on both input and output observations, given that we are in non-terminal node (1). This is given by

$$h_{1|1} = P(z_{11}|z_1,\boldsymbol{x},\boldsymbol{y}) = \frac{P(z_{11}|z_1,\boldsymbol{x})P(\boldsymbol{y}|\boldsymbol{x},z_{11},z_1,\Theta_{11})}{\sum_{j=1}^{b_1} P(z_{1j}|z_1\boldsymbol{x})P(\boldsymbol{y}|\boldsymbol{x},z_{1j},z_1,\Theta_{1j})},$$

where $P(\boldsymbol{y}|\boldsymbol{x},z_{11},z_1,\Theta_{11})$ is the probability of generating the correct output vector $\boldsymbol{y}$ from EN$(1,1)$ given the input $\boldsymbol{x}$. For 1-out-of-N classification, this is given by

$$P(\boldsymbol{y}|\boldsymbol{x},z_{11},z_1,\Theta_{11}) \;=\; \exp\left(\sum_{k=1}^{N} y_k \log \mu_{11k}\right) = \mu_{11n}$$

where μ_{11k} is the output for class k from EN$(1,1)$ and n is the correct class. In a similar way, the conditional probability of node (1) is given by

$$h_1 = P(z_1,\boldsymbol{x},\boldsymbol{y}) = \frac{P(z_1|\boldsymbol{x})P(\boldsymbol{y}|\boldsymbol{x},z_1)}{\sum_{i=1}^{b_1} P(z_i|\boldsymbol{x})P(\boldsymbol{y}|\boldsymbol{x},z_i)}.$$

For node $(1,1)$ the *joint* posterior probability, h_{11} is the product of the joint posterior probability of node (1) and the conditional posterior probability of node $(1,1)$. In a deeper architecture, the joint posterior probabilities are recursively computed by multiplying the conditional posterior probabilities along a path from the root node (0) to the node in question.

The M step reduces to a set of independent *weighted* maximum likelihood problems for the experts and the gates. Thus the *weight* for GN(1), at time t, is the joint posterior probability of this node, $h_1^{(t)}$, and the *weight* for EN$(1,1)$ is the joint posterior probability of this node, $h_{11}^{(t)}$. The target outputs for the gating networks are the conditional posterior probabilities of the node in question, so that the targets for GN(1) at time t are $h_{1|1}^{(t)}$ and $h_{2|1}^{(t)}$ for outputs 1 and 2 respectively.

Once the maximum likelihood problems of the M-step have been completed, the E-step is repeated, computing a new set of posteriors $\{H\}$ for all times t which become the new weights for the M-step.

Solving the M-Step

Since each EN and GN is a simple network with a single layer of weights, we may solve the maximum likelihood problems relatively easily. We update the parameter vectors for each output of the networks independently, given the *Generalised Linear* [6] assumption that the outputs are independent. The simplest method is to use gradient ascent of the likelihood, which for parameter vector θ_i^m at iteration m for output i of an EN reduces to

$$\theta_i^{m+1} = \theta_i^m + \lambda \frac{1}{\sum_{t=1}^{T} h^{(t)}} \sum_{t=1}^{T} h^{(t)}\boldsymbol{x}^{(t)}(y_i^{(t)} - \mu_i^{(t)})$$

where $y_i^{(t)}$ is the target for class i, $\mu_i^{(t)}$ is the i^{th} output of the EN, $h^{(t)}$ is the *weight* at time t, T is the total time, and λ is a learning rate. These equations are the same for the gating networks, with the output targets $\{y_i^{(t)}\}$ replaced by the conditional posterior probabilities of the node in question.

An alternative maximisation method, and the one adopted in this paper, is to use the Hessian or second derivative of the likelihood with respect to the parameter vectors:

$$\theta_i^{m+1} = \theta_i^m + \lambda \left(\sum_{t=1}^{T} h^{(t)} \boldsymbol{x}^{(t)} \mu_i^{(t)} (1 - \mu_i^{(t)}) \boldsymbol{x}^{(t)^T} \right)^{-1} \left(\sum_{t=1}^{T} h^{(t)} \boldsymbol{x}^{(t)} (y_i^{(t)} - \mu_i^{(t)}) \right), \qquad (1)$$

where λ is once again a learning rate, which has typical values in the range 0.4 to 1.0. This method is equivalent to the Iteratively Reweighted Least Squares algorithm (IRLS) of Generalised Linear Models [6].

Implementation Issues

Variation in M-Step Iterations. Although the basic EM algorithm dictates that the M step should be iterated until convergence, the *Generalised* EM algorithm (GEM) relaxes this constraint, requiring only an increase in the likelihood in the M-step. By reducing the number of M-step iterations we can reduce the overall computation. The power of the EM algorithm lies in the E-step which repeatedly computes new weights based on the previous M steps. In our experiments the number of M step iterations was typically set to between 1 and 3.

Learning rates. The IRLS algorithm in common with conventional gradient descent algorithms, is sensitive to learning rates. Learning rates that are too large give instability, manifested in step sizes that lead to an decrease in the overall network likelihood. In practice we found that a learning rate of 0.4 for both experts and gates gave a good balance between learning speed and stability, although rates of 0.8 have proved stable with some initial conditions.

Saturation of expert and gating network outputs. If the output $\mu_i^{(t)}$ in Equation (1) of any of the networks becomes near to either 1 or 0, or if the weight $h^{(t)}$ is near 0, then the addition to the Hessian matrix for output i of that network at time t will be very small. If this occurs for a large majority of the training set, the Hessian will become singular and impossible to invert accurately. The solution to this problem is to use threshold values for the outputs of 0.9999 and 0.0001 and a floor for the weights of 0.0001. In practice, these have to be tuned to prevent instability but have no significant effect on accuracy until set to around 0.9 and 0.1.

Choice of initial parameter values. Two strategies are used to initialise the network. The first is to start all parameter vectors of experts and gates at zero and give each gate output a ‘kick’ so that the experts begin to separate in the input space and compute different outputs. The second is to initialise all parameter vectors to random values, in a range $-r$ to $+r$. Typically, $0.1 \leq r \leq 3$. An alternative is to use random weights for the expert parameters and zero initial weights for the gates, with the choice of strategy varying with the problem. In our experiments we found that the second option gave the quickest results which were most free from local maxima, whilst the first and third options gave solutions which were drawn to local maxima or failed to separate the experts at all.

SIMULATIONS

In this paper we follow the work of [7] in using strict methods when reporting learning speeds and network performance. In particular we use a 40-20-40 threshold criterion which dictates that an output is only correct if it is greater than 0.6. We define an *epoch* as one pass through the training set. Thus, one EM cycle may consist of many epochs, depending on the number of iterations performed in the M-step.

N-Input Parity

The task of the N-input parity problem is to compute the odd parity of N binary inputs. The network must compute a 'one' if the input has an odd number of 'one' bits in the input and a 'zero' if there are an even number. The special case of 2-input parity is the Exclusive Or function (XOR) which was shown to be impossible for simple single layer networks to approximate [8]. In this paper we show that the HME can solve this problem efficiently using only three single layer networks, in the form of one GN and two ENs. We also describe solutions for 3 to 8 input parity, with learning times faster than conventional feed-forward networks. The performance of the HME on the XOR problem using a varying degree of test thresholding and averaging over 100 trials per threshold is shown in Table 1. These results were obtained using a HME(1:2) with a total of 9 parameters. By way of comparison, conventional feed forward networks, using a 2-2-1 structure can solve this problem at the 0.6 threshold in an average of 19 epochs of quickprop [7]. Using the delta-bar-delta rule, an average training time of 250.4 epochs has been reported [9]. Using the HME, we reach the 0.6 threshold in an average of 2.76 epochs, and the 0.99 threshold in an average of 11.5 epochs. Of all these trials of the HME on the XOR problem, none failed to converge or had to be restarted. The results on the N-input Parity problem are shown in Table 2. By way of comparison, the best performance reported on 8-input Parity by a 8-16-1 back-propagation network is 2000 epochs of standard back-propagation [10] and 172 epochs of quickprop. The table shows the average results over 50 trials. The number of tests which failed to converge to the correct solution is shown as % NC. Using the HME, N-input parity requires at least N experts. However, in 'tight' networks with around *N* experts, there is an increased chance of local maxima. This effect may be seen in Figure 2 which shows the effect of different initial conditions for a HME(1:4) on the XOR problem. Since we may solve the XOR problem using only 2 experts, this network is over-specified and includes redundancy. The solutions in the figure differ in the distribution of the data between the experts. Sub-figure 2(a) is a solution in which the data is shared evenly between 2 experts with the remaining 2 experts inactive. In 2(b) the data is distributed again between 2 experts but with

Threshold	Min Epochs	Max Epochs	Average Epochs	Standard Deviation
0.5	1	6	2.35	1.02
0.6	2	6	2.76	0.971
0.8	2	6	3.32	0.882
0.9	3	6	4.14	0.757
0.99	8	24	11.5	4.27

Table 1: Results for the HME on the XOR problem.

N	Parameters	Architecture	Min Epochs	Max Epochs	Average Epochs	% NC
3	12	(1:3)	3	11	4.85	0
4	18	(2:2,2)	6	13	10.4	20
5	51	(2:3,3)	11	26	16.2	10
6	60	(4:2,2,2,2)	12	29	18.8	20
7	111	(5:2,2,2,2,2)	14	26	23.5	10
8	51	(2:2,4)	50	102	42	90
8	111	(5:2,2,2,2,2)	15	57	34	50
8	210	(6:2,2,2,2,2,2)	13	56	36	33

Table 2: Performance of the HME on the N-Input Parity problem.

3/4 of the data in one expert and 1/4 in the other. In 2(c) the data is distributed over 3 experts with 1/2 going to one expert and the remaining 1/2 shared between the remaining 2 experts. In 2(d) the data is distributed evenly over 4 experts. It is clear that 2 (b) is an unsatisfactory solution which would not give good generalisation, unlike (a), (c) or (d). In a series of 35 trials for HME(1:2), solution (a) occurred in 26 cases while (b) occurred in 9 cases. By using the HME(1:4), solution (c) occurred 23 times, (a) 8 times, (d) 3 times and (b) only once. Thus, we have reduced the probability of solution (b) occuring by adding the extra 2 experts. For larger values of N, behaviour of this sort may lead to local maxima which give us non-convergent solutions. The net result of this is that we get many more non-convergent solutions with tight networks, although there is a large advantage in terms of computation when using such a network. By relaxing the network and using more levels, thus introducing redundancy, we create many more possible successful configurations, as seen by relaxing the XOR problem to using 4 experts instead of 2. Therefore the number of non-convergent solutions is reduced, and those that do occur represent states where only a small number of points remain misclassified. This effect may be seen for 8-input Parity in Table 2. By increasing the network depth, thus increasing the number of terminal nodes, we reduce the percentage of non-convergent solutions.

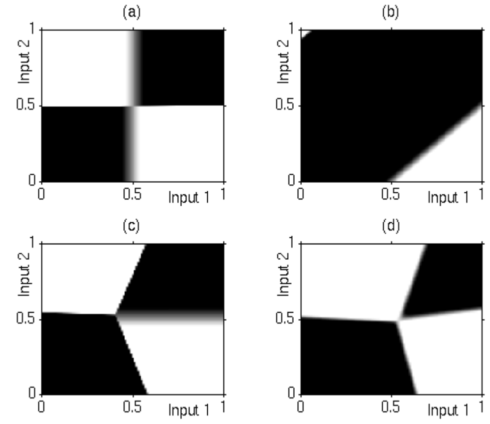


Figure 2: The effect of different initial conditions on a HME(1:4) for the XOR problem.

The Two Spirals Problem

The aim of the two-spirals problem is to train a network to discriminate between two spirals in the 2-D plane. Each spiral has 97 points and coils three times around the origin and around the other spiral, without overlapping. The learning set and the evolution of the output of a binary branching HME with 10 levels is shown in Figure 3. The points in the test set are offset vertically from the points in the learning set by 0.1. The best solutions to the spirals problem have been obtained using Cascade Correlation [11]. This is capable of approximating the problem in 1700 epochs using around 140 parameters. Back-propagation networks have been used to solve the problem [12] using a 2-5-5-1 network with shortcut connections between layers in $20,000$ epochs using conventional gradient descent with momentum and $8,000$ epochs using quickprop, using a similar number of parameters. Using a conventional 2-5-5-1 network without shortcuts took $60,000$ epochs of quickprop.

Depth of Tree	Number of Parameters	Training Set Correct / 194	Testing Set Correct / 194	M-step Iterations	Total Epochs
10	3102	187	184	3	135
10	3102	185	184	1	140
5	111	161	159	1	30

Table 3: Results for the Two-Spirals Problem using binary branching HMEs.

The results in Table 3 demonstrate that the HME is capable of solving the two-spirals problem to a high degree of accuracy. Although the experiments performed have not resulted in a complete solution, the number of training epochs for the HME on this problem are an order of magnitude less than Cascade Correlation networks and two orders of magnitude less than conventional feed-forward back-propagation networks. We suspect that the non-convergence of the HME is due to similar effects as those proposed for the Parity problem. In terms of numbers of parameters, the HME may appear to be using far more, since for a depth of 10 and common branching factor of 2 there are 3102 parameters. This is misleading, however, since the number of terminal nodes which actually remain active is a small fraction of the total number of terminal nodes present.

CONCLUSIONS

We have described the application of the HME to classification and presented a number of results on standard benchmarks. In common with the performance of the HME on regression problems, we have found that it requires fewer epochs to learn classification problems than conventional feed-forward networks. There are however, a number of problems associated with the learning algorithm, including numerical instabilities caused by the 2nd order M-step update and the existence of local maxima within the solution sets. We have described solutions to these problems, such as the use of thresholding of outputs and weights in the M-step, choice of learning rate and initial conditions to avoid instabilities and local maxima. Future directions for this work will focus on removing the need for matrix inversion by using some form of approximation to the inverse Hessian in (1). The use of fast gradient descent methods such as quickprop [7] would move the HME closer to true connectionist methods and reduce the computational load of the M-step.

In addition we have described how the use of redundancy in the HME may reduce the chance of local maxima. In these networks, the redundant experts are typically inactive after a few epochs, which suggests that they could be 'pruned' using similar techniques to those developed for CART [3] . Alternatively we may start with a small network of, say 2 experts and grow the tree using CART principles. We anticipate that the use of such ideas will improve the performance of the HME in terms of speed and accuracy and allow us to extend the applications of classification HMEs to real world problems.

Acknowledgements

Steve Waterhouse is funded by a partnership agreement with Waterhouse Associates. Tony Robinson is supported by an EPSRC Advanced Research Fellowship.

REFERENCES

[1] M. I. Jordan and R. A. Jacobs, "Hierarchical Mixtures of Experts and the EM algorithm," *Neural Computation*, vol. 6, pp. 181–214, 1994.

[2] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society, Series B*, vol. 39, pp. 1–38, June 1977.

[3] L. Breiman, J. Friedman, R. Olshen, and C. J. Stone, *Classification and Regression Trees.* Wadswoth and Brooks/Cole, 1984.

[4] D. H. Wolpert, "Stacked generalization," Tech. Rep. LA-UR-90-3460, The Santa Fe Institute, 1660 Old Pecos Trail, Suite A, Santa Fe, NM, 87501, 1993.

[5] J. S. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in *Neurocomputing: Algorithms, Architectures and Applicatations* (F. Fougelman-Soulie and J. Hérault, eds.), pp. 227–236, Springer-Verlag, 1989.

[6] P. McCullagh and J. A. Nelder, *Generalized Linear Models.* London: Chapman and Hall, 1989.

[7] S. E. Fahlman, "Faster-learning variations on back-propagation: An empirical study," in *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann, 1988.

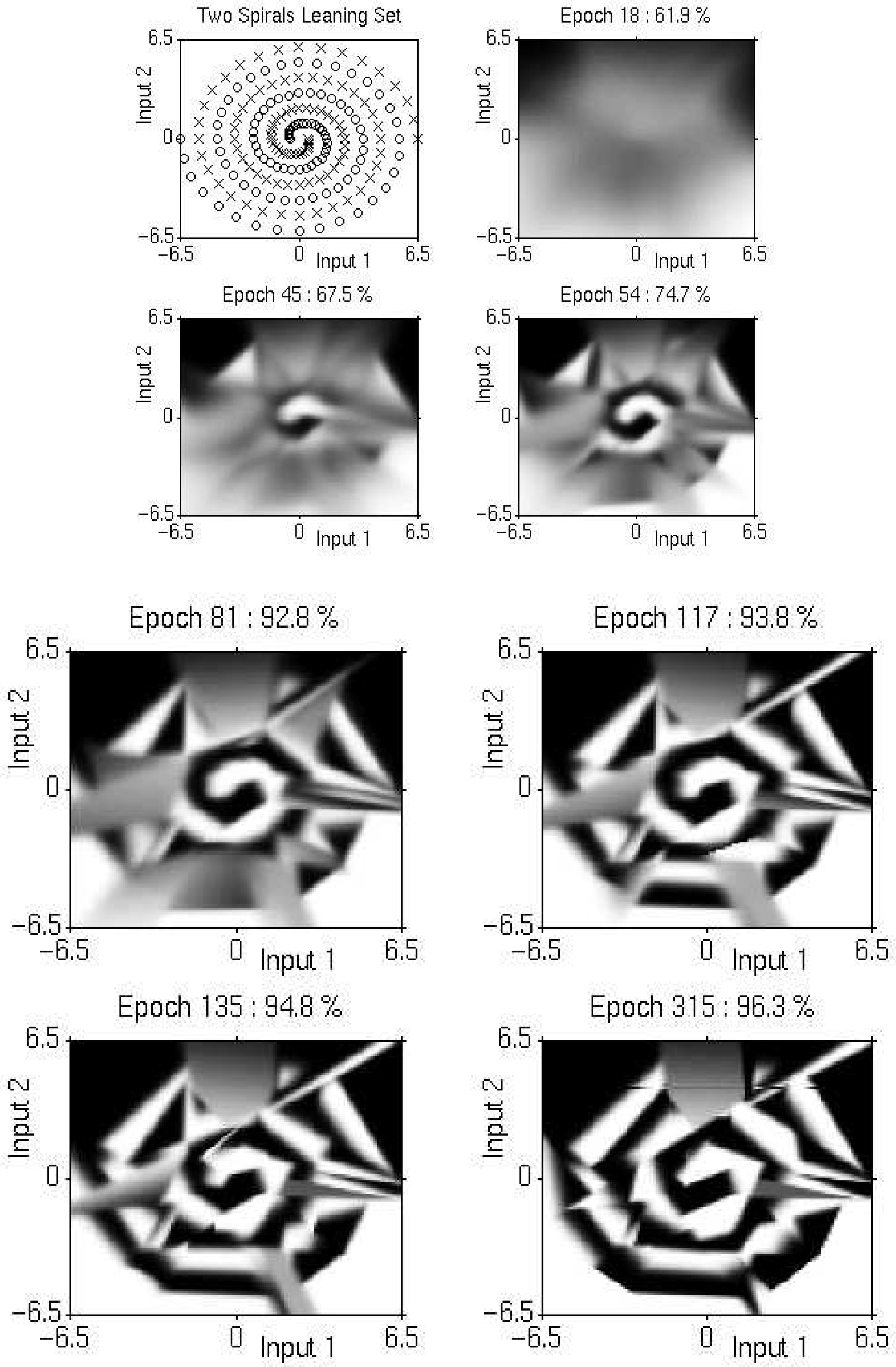
[8] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry.* Cambridge, MA: MIT Press, 1969.

[9] R. A. Jacobs, "Increased rates of convergence through learning rate adaptation," *Neural Networks*, vol. 1, pp. 295–307, 1988.

[10] G. Tesauro and R. Janssens, "Scaling relationships in back-propagation learning: Dependence on predicate order," Tech. Rep. CCSR-88-1, Center for Complex Systems Research, University of Illinois at Urbana Champagne, 1988.

[11] S. E. Fahlman and C. Lebiere, "The Cascade-Correlation learning architecture," Tech. Rep. CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, 1990.

[12] K. J. Lang and M. J. Witbrock, "Learning to tell two spirals apart," in *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann, 1988.


Two Spirals Leaning Set
Epoch 18 : 61.9 %
Epoch 45 : 67.5 %
Epoch 54 : 74.7 %
Epoch 81 : 92.8 %
Epoch 117 : 93.8 %
Epoch 135 : 94.8 %
Epoch 315 : 96.3 %
Input 1
Input 2
6.5
0
−6.5