Ensemble Learning using Decorrelated Neural Networks

Bruce E. Rosen
Computer Science Division
University of Texas at San Antonio
San Antonio, TX 78230

Abstract

We describe a decorrelation network training method for improving the quality of regression learning in "ensemble" neural networks that are composed of linear combinations of individual neural networks. In this method, individual networks are trained by backpropagation to not only reproduce a desired output, but also to have their errors be linearly decorrelated with the other networks. Outputs from the individual networks are then linearly combined to produce the output of the ensemble network.

We demonstrate the performances of decorrelated network training on learning the "3 Parity" logic function, a noisy sine function, and a one dimensional nonlinear function, and compare the results with the ensemble networks composed of independently trained individual networks (without decorrelation training). Empirical results show that when individual networks are forced to be decorrelated with one another the resulting ensemble neural networks have lower mean squared errors than the ensemble networks having independently trained individual networks. This method is particularly applicable when there is insufficient data to train each individual network on disjoint subsets of training patterns.

1 Introduction

Back-propagation networks are often of particular interest in many applications [1] because they can approximate any square integrable function to any desired degree of accuracy based on an arbitrary finite size training set. Most current research has sought to improve network processing speed through fast training algorithms [2], or to reduce generalization error using pruning methods [3] [4].

Recently there has developed a mounting interest in examining ensemble networks, where each ensemble is composed of several individually trained neural networks. The outputs of the individual networks are combined to produce the output of the ensemble network. The individual networks are usually combined by simple averaging or by linear regression, but other methods such as using the Order Statistics [5] can also be used. We focus on ensemble

networks that are linear combinations of individually trained networks. Perrone [6] and Hanshem [7] have shown that these ensemble networks often outperform their individual networks.

Ensemble neural networks are only useful when the individual networks disagree in their predictions [8]. There are several approaches for creating disagreeing networks. One approach is to train several individual networks separately and hope that their predictions will be some different. A second approach is to use different activation function or architecture in each individual network. A third approach is to train the individual networks on different subsamples from the original training set [9, 10]. However, the approach taken here is to train the individual networks not only to reduce their approximation errors and but also to reduce the correlations of individual networks errors.

2 Ensemble Networks

Supervised neural network learning algorithms are designed to learn the functional relationship of a set of patterns. The patterns are usually composed of an input vector $\vec{x}$ and a desired output scalar value y. The training set is a set of N patterns $\{(\vec{x}_1, y_1)...(\vec{x}_N, y_N)\}$ with the pth pattern defined by some unknown functional relationship:

$$y_p = g(\vec{x}_p) + \epsilon,$$

where g is a regression function, and ϵ is some mean zero additive noise with finite variance σ^2. The learning algorithm trains the network to minimize the error criteria which is often defined as the sum of squared errors:

$$E_f = \sum_{p=1}^{N} (y_p - f(\vec{x}_p))^2, \tag{1}$$

where $f(\vec{x}_p)$ is the network output given the pth training pattern. This error can be restated in terms of the network's bias and variance [11]:

$$E_f = E\{\mathrm{Var}_f + \mathrm{Bias}_f^2\} + \sigma^2, \tag{2}$$

where E$\{\}$ is the expectation operator, $\mathrm{Var}_f = E\{(f(\vec{x}) - E\{f(\vec{x})\})^2\}$ is the variance, and $\mathrm{Bias}_f = E\{f(\vec{x})\} - g(\vec{x})$ is the bias.

The output of a *basic* ensemble network is the average of the M network outputs

$$f_{\text{ens}}(\vec{x}) = \frac{1}{M}\sum_{j=1}^{M} f_j(\vec{x}), \tag{3}$$

where $f_{\text{ens}}(\vec{x})$ is the output of the basic ensemble network, and $f_j(\vec{x})$ produces the output of the jth network. The output of a *general* ensemble network is the weighted average of the individual network's outputs

$$f_{\text{ens}}(\vec{x}) = \sum_{j} \alpha_j f_j(\vec{x}), \tag{4}$$

where α_j is the weight associated with the jth network's output. After the individual networks are trained, the ensemble network weights α are found by singular value decomposition, by the delta rule, or by other methods.

Assuming that all networks learn from the same set of training data, then the generalization error from a basic ensemble network is E_{ens}, defined by the following theorem (from Ueda [12]):

Theorem

Let E_{ens} be the generalization error of the basic ensemble composed of M individual networks. Then the ensemble error is defined as

$$E_{\text{ens}} = E\{\frac{1}{M}\overline{\text{Var}} + (1 - \frac{1}{M})\overline{\text{Cov}} + \overline{\text{Bias}}^2\} + \sigma^2, \tag{5}$$

where

$$\begin{aligned}
\overline{\text{Var}} &= \frac{1}{M}\sum_{j=1}^{M} \text{Var}_j, \\
\overline{\text{Cov}} &= \frac{1}{M(M-1)}\sum_{j,k=1,k\neq j}^{M} \text{Cov}_{i,j}, \text{ and} \\
\overline{\text{Bias}} &= \frac{1}{M}\sum_{j=1}^{M} \text{Bias}_j.
\end{aligned} \tag{6}$$

Here Var_i, and Bias_i refer to the variance and bias of network i, and $\text{Cov}_{i,j}$ refers to the covariance of networks i and j. The proof of the above theorem can be found in [12].

If all the networks have the same architecture and structure then the expected variance and biases of the networks

will be the same, then

$$E_{\text{ens}} = E\{\frac{1}{M}\text{Var} + (1 - \frac{1}{M})\text{Cov} + \text{Bias}^2\} + \sigma^2.$$

One way to minimize E_{ens} in Equations (5) is to reduce $E\{\overline{\text{Cov}}\}$ without increasing $\overline{\text{Var}}$ or $\overline{\text{Bias}}^2$. Since the error of the basic ensemble network is affected by the covariance of network errors, negative error correlation contributes to a decrease in the generalization error. A similar analysis can be applied for general ensemble networks [12].

3 Decorrelation Networks

When combining multiple back-propagation networks each individual network is typically trained independently to minimize Equation (1) [6]. Colinearity problems, which reduce the effectiveness of Equations (3) and (4), can occur when the individual network errors are correlated. Bunn [13] [14] states that when combining forecasts, no major gains can be expected if there are large positive correlations between the forecasting errors. And Hashem [7] shows that correlations between network outputs may make optimal linear combinations of networks prone to colinearity problems.

To illustrate the network correlation problem, Figure 1a shows two networks trying to approximate a desired output surface. After learning, both networks approximate the surface in the same manner leading to significant correlations in the network errors. Generally, both networks simultaneously overestimate or underestimate the desired output. The network outputs are highly correlated potentially leading to severe colinearity and reducing the robustness of the ensemble network [7]. To mitigate this potential colinearity problem, Equation (1) is modified by adding a decorrelation penalty to it. The individual networks attempt to not only minimize the error between the target and their output, but also to decorrelate their errors with those from previously trained networks. The new error function for an individual network j is:

$$E_j = \sum_{p=1}^{N} \left[\left(y_p - f_j(\vec{x}_p)\right)^2 + \sum_{i=1}^{j-1} \lambda(t) d(i,j) P(\vec{x}_p, y_p, f_i, f_j) \right], \tag{7}$$

where p is the pth pattern, $\lambda(t)$ is a (possibly) time dependent scaling function, d is an indicator function for decorrelation between networks i and j, and P is a correlation penalty function.

The first term in Equation (7) is used to minimize the errors between the network's output and its target output.

The second term in the equation is a penalty term similar to the weight decay regularization term used in [3]. The intent of this term is not to smooth the network output or to reduce the magnitude or the number of weights [15], but to decrease the correlation between the individual networks.

Figure 1 illustrates the rationale for decorrelating networks. In Figure 1a two networks have similar input-output function approximations and have errors that are highly correlated. In Figure 1b the two networks have different input-output function approximations and have errors that are uncorrelated. Linearly combining the networks in Figure 1b improves the regression approximation in one area of the mapping while it degrades the regression approximation in another. Figure 1c shows two networks with different input-output function approximations and negatively correlated errors. Combining the two networks by simple averaging will improve the ensemble function approximation. Thus, ensemble networks composed of negatively correlated networks can outperform ensemble networks composed of uncorrelated networks. The gain in performance is due to increased cancellation of errors within negatively correlated networks as opposed to errors in uncorrelated networks. Therefore it is advantageous to choose the elements of penalty term given in Equation (7) so as to encourage individual networks to be decorrelated.

One problem with decorrelated ensemble networks is that, even though the individual networks may be highly decorrelated with one another, the ensemble network performance (as defined in Equation (1)) may be poor. This condition can occur when the decorrelation penalty is to large. Although the covariance between the individual networks will be reduced, it will be at the expense of an increase in individual network errors. In these circumstances, it is advisable to anneal the decorrelation penalty to alleviate the problem. Initially, the penalty forces new networks to develop different functional mappings. As the penalty is annealed, the networks focus more on reducing the pattern errors than on being decorrelated.

Figures 2a and 2b describe this process. In the figures, the solid curve represents the desired functional output for every input in the single input domain. The dashed line represents the first network's function approximation of the curve. The dotted line shows the second network's approximation of the desired curve after several "decorrelation" training epochs. The second network provides an alternative functional mapping to the desired output and is negatively correlated with the first network. However, the second network's error is large. After annealing the decorrelation penalty, the correlation between networks is still negative, but the second network provides a better approximation to the desired network.

4 Decorrelation Functions

The penalty term in Equation (7) consists of three functions: a correlation penalty function, an indicator function, and a scaling function. The correlation penalty function P is the product of the jth and ith network error:

$$P(\vec{x}, y, f_i, f_j) = (y - f_i(\vec{x}))(y - f_j(\vec{x})). \tag{8}$$

This function is positive only when the network errors are positively correlated and have the same sign. The more decorrelated two individual networks are, the more negative P becomes.

The indicator function d specifies which individual networks are to be decorrelated. For example, to penalize an individual network for being correlated with the previously trained network, the indicator function is

$$d(i,j) = \begin{cases} 1 & \text{if } i = j-1 \\ 0 & \text{otherwise} \end{cases}. \tag{9}$$

To allow alternate networks to be trained independently of one another yet decorrelate pairs of networks, the indicator function can be defined as

$$d(i,j) = \begin{cases} 1 & \text{if } i = j-1 \text{ and } i \text{ is even} \\ 0 & \text{otherwise} \end{cases}. \tag{10}$$

The scaling function $\lambda(t)$ is either constant [1] or is time dependent.

5 Methodology

The performances of regular ensemble networks (trained to minimize Equation (1)) and decorrelation ensemble networks (trained to be decorrelated by minimizing Equation (7)) were compared on three network learning tasks. The first task was for the networks to learn the "3 Parity" logic function, the second task was for the networks to learn a noisy sine function, and the third task involved learning a noisy nonlinear function from a nonuniformly distributed pattern set. In each task, the ensemble networks were composed of linear combinations of back-propagation trained

[1] typically determined by cross validation

networks.

5.1 Learning the Three Parity Function

In the "3 Parity" logic problem, the ensemble networks were composed of several individual networks, each trained to approximate the three parity logic function. All individual network architectures consisted of three input units, a single hidden unit, and a single output unit. This limited configuration prevented the individual networks from completely learning the task and restricted any individual network to a minimum error of 0.1875.

Eight networks were first trained by back-propagation with different random number generator seeds. Each ensemble network consisted of between two and eight individual networks, whose outputs were linearly combined to produce the ensemble network output. The ensemble network weights were learned using the delta rule to minimize Equation (1). During learning, each network training cycle was terminated after 1000 training epochs, or when the gradient vanished. All back-propagation learning algorithms used the Fletcher-Reeves version for conjugate gradient descent binary line search and parabolic interpolation [16]. The line search and parabolic interpolation aspects of the algorithm do not require error gradients to be calculated, so there are fewer backward passes than forward passes. Additionally, parameter tuning is not required since there are no gain, momentum or additional terms to adjust. Overall, we have found this method to be faster than back-propagation with momentum or other extended methods such as Jacob's Delta-Bar-Delta [2] or Fahlman's Quickprop [17] for training most data sets.

We compared the performances of Equation (1) with Equation (7). For the decorrelation networks, different combinations of the indicator functions in Equations (9) and (10) were used with $\lambda(t)$ either constant or decaying inversely with each training pass. Each method was tested 50 times using different initial random seeds, and hence different initial weights. However, the same initial random seed was used for corresponding combinations.

Figure 3 shows the average performances (over 50 trials) of three ensemble networks trained on the "3 Parity" task plotted against the number of individual networks in each ensemble network. One set of ensemble networks, *No Penalty*, was trained without decorrelation penalties, while the remaining two sets of ensemble networks, *Penalty 1* and *Penalty 2*, were trained with decorrelation penalties. In the figure, the *No Penalty* performances were from the individual networks trained independently using Equation (1). The *Penalty 1* performances were from the ensemble networks whose individual networks were trained with $\lambda(t)$ constant but were penalized for being correlated with

the previous network (9). The *Penalty 2* performances were from ensemble networks. In each ensemble network, the even numbered individual networks were decorrelated only with the network immediately preceding it. The indicator function was Equation (10) and $\lambda(t)$ was set to the constant $1/i$, where i is the ith network. The best performances for the three parity problem were those from the *Penalty 2* ensemble networks.

It is interesting to note that this problem belongs to the 0-1 classification domain rather than the continuous regression domain. In the classification domain, the network outputs should never exceed the bounds of the desired outputs (0,1) (or (-1,1) for bipolar outputs). Therefore the decorrelation penalty of Equation (8) will always be nonnegative. Given two patterns with equal expected network errors but with different error correlations, Equation (7) will emphasize the minimization of pattern errors associated with larger correlations. For example, during the training of network j if $E_{f_i(\vec{x}_1)} = E_{f_j(\vec{x}_1)} = 0.5$, $E_{f_i(\vec{x}_2)} = 0.1$, and $E_{f_j(\vec{x}_2)} = 0.9$, then $E_{f_j(\vec{x}_1)}$ will contribute more to error E_j in Equation (7) than $E_{f_j(\vec{x}_2)}$. Hence, the network j will seek more to minimize the error associated with pattern $\vec{x}_1$ than $\vec{x}_2$.

5.2 Learning a Noisy Sine Function

The second learning problem was for the ensemble networks to approximate the noisy sine function given by

$$y_p = 0.5 * \sin(\pi x_p/50) + \eta,$$

where $x_p = 0...3.5$ and η is mean zero Gaussian noise with variance $\sigma^2 = 0.02$. Eighty-eight uniformly distributed patterns were used in total. In these experiments, all ensemble networks were composed of two individual networks. The single input, single output individual networks had two hidden layers and seven hidden units per layer, and were trained for 15000 epochs. One hundred and fifty learning runs were performed for the ensemble networks trained 1) independently, and 2) using Equation (7), with $d(i,j)$ defined by Equation (9), and $\lambda(t) = 100/(N * t)$, where t is the number of training epochs.

Table 1 shows the results of these simulations. The *No Penalty* column shows the performance results when the individual networks were independently trained and combined to form the ensemble 1-2, (i.e. the outputs from individual networks 1 and 2 were used as the inputs to ensemble network 1-2.) The *Penalty* column shows the network performances when the individual networks were trained to be decorrelated. Even though the mean performance of

the second (individual) decorrelation networks were larger than those of all other networks, the mean performance of the ensemble decorrelation networks was substantially lower than those of all other networks.

5.3 Learning a Noisy Nonlinear Function

The last experimental problem was to learn a noisy nonlinear function from a set of nonuniformly distributed patterns. The distribution of the scalar pattern input x was defined as the mixture of two Gaussians given by

$$P(x) = d_1 N(\mu_1, \sigma_1^2) + d_2 N(\mu_1, \sigma_2^2), \tag{11}$$

where $N(\mu, \sigma^2)$ is a Gaussian distributed random number generated with mean μ and variance σ^2. In our simulations, $d_1 = 0.25$, $\mu_1 =$ -2, $\sigma_1 =$1, $d_2 = 0.75$, $\mu_2 = 2$, and $\sigma_2 =$2. The desired noisy nonlinear output was produced by utilizing the nonlinear function $f(x)$ given by

$$f(x) \quad = \quad 0.5x + 1.5e^{(-.3x)} - 2e^{(-.4*x*x)} + 0.5\,\sin(4x) + \eta(.1 + |x-1|/5),$$

where η is mean zero Gaussian distributed noise with variance 1. The sample distribution is shown in Figure 4. The pattern data is not uniformly distributed about the domain, and the variance of the noise is clearly seen to be minimized at $x = 1$.

Five individual networks were trained to predict the values of a noisy nonlinear function. The ith individual network contained one input unit and i hidden units. There were no direct input to output links, and each network was trained for 1000 epochs before finishing. Table 2 shows the average testing prediction errors (over ten trials) for the five individual networks when trained with Equation (1), and when trained with the added penalties given in Equation (7) with λ constant. Each ensemble network was composed of two to five individual networks. Also shown in the table are the performances of the four ensemble networks. Ensemble network "1-i" indicates that the ensemble network was composed of individual networks 1 through i, i.e. the outputs from networks 1 through i were used as the inputs to ensemble network "1-i".

The *No Penalty* column shows the performance results when the individual networks were independently trained and combined to form the ensemble networks. The *Penalty* column shows the network performances when the

individual networks were trained to be decorrelated. The largest effect of the decorrelation appears when networks one and two are combined. The improvement continues, however, as further decorrelated networks are added. In the nonlinear function approximation simulations, the ensemble decorrelation network performances were better when their individual networks were trained to be decorrelated.

6 Conclusions

We have described a method for improving the performances of ensemble neural networks by training their individual networks to be decorrelated with one another. Our experimental results have shown that the errors from linear combinations of individual networks can be reduced when the individual networks learn to be decorrelated. In contrast to the Cascade Correlation learning architecture [18] which attempts to create new hidden units and tries to maximize the magnitude of the correlation between a new unit's output and the residual error, decorrelation networks attempt to minimize the correlation of errors between the individual networks. Our results show that the addition of a decorrelation penalty term in the training process can enhance the performance of linearly combined back-propagation networks.

Future investigations will examine: 1) Decorrelating networks that have different activation functions. The decorrelation approach described here is not constrained on any particular activation function. Hence, different activation functions [19] could also be used (e.g. radial basis or spline functions) to decorrelate the networks. 2) Choosing appropriate indicator functions $d(i,j)$. The indicator function can affect the parallelization of the algorithms, e.g. Equation (9) may be less parallelizable than Equation (10), and 3) Creating new ensemble networks by linearly combining an (already trained) ensemble network with individual networks that are trained with a decorrelated penalty, i.e. each penalty function is of the form:

$$E_j = \sum_{p=1}^{N} \left(y_p - f_j(\vec{x}_p)\right)^2 + \sum_{i=1}^{j-1} \lambda(t) d(j, m_i) P(\vec{x}_p, y_p, f_j, f_{m_i}),$$

where m_i represents the ith ensemble network. This approach should be even more effective at minimizing the ensemble network's error.

Acknowledgments

We would like to thank Dmitry Gokhman, Bob Hiromoto, Tom Bylander, and Tina Chang for helpful discussions.

References

[1] A. Lapedes and R. Farber. How neural nets learn. In *Neural Information Processing Systems*. AIP Press, 1988.

[2] R. A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 4(1), 1989.

[3] A. S. Weigend, D. E. Rummelhart, and B. A. Huberman. Back-propagation, weight elimination, and time series prediction. In D.S. Touretzky et al., editor, *Proceedings of the 1990 Connectionist Models Summer School*. Morgan Kaufmann, 1990.

[4] A. S. Weigend, D. E. Rummelhart, and B. A. Huberman. Generalization by weight elimination with application to forecasting. In R. Lippmann et al., editor, *Advances in Neural Information Processing Systems*, volume 3. Morgan Kaufmann, 1991.

[5] Kangan Tumer and Joydeep Ghosh. Order statistics combiners for neural classifiers. In *Proceedings of the World Congress on Neural Networks*, pages I31–34. INNS Press, Washington, D.C., July 1995.

[6] Michael P. Perrone. Improving regression estimation: Averaging methods for variance reduction with extensions to general convex measure optimization. Technical Report Ph.D. Dissertation, Brown University, May 1993.

[7] Sherif Hashem. Optimal linear combinations of neural networks. Technical Report Ph.D. Dissertation, Purdue University, December 1993.

[8] Anders Krogh and Jesper Vedelsby. Neural network ensembles, cross validation, and active learning. In *Neural Information Processing Systems*, volume 7. MIT Press, 1995.

[9] Bambang Parmanto, Paul W. Munro, and Howard R. Doyle. Improving committee diagnosis with resampling techniques. In D. Touretzky, Michael Mozer, and Michael Hasselmo, editors, *Neural Information Processing Systems*, volume 8. MIT Press, 1996.

[10] Peter Sollich and Anders Krogh. Learning with ensembles: How over-fitting can be useful. In *Neural Information Processing Systems*, volume 7. MIT Press, 1995.

[11] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58, 1992.

[12] N. Ueda and R. Nakano. Statistical analysis of the generalization error of ensemble estimators. *(submitted to) Neural Computation*, 1995.

[13] D. W. Bunn. Forecasting with more than one model. *Journal of Forecasting*, 8:161–166, 1989.

[14] D. W. Bunn. Statistical efficiency in the linear combination of forecasts. *International Journal of Forecasting*, 1:151–163, 1985.

[15] W. L. Buntine and A. S. Weigend. Bayesian back-propagation. *Complex Systems*, 5, 1991.

[16] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1988.

[17] S. E. Fahlman. Faster-learning variations on back-propagation: An empirical study. In *Proceedings of the 1988 Connectionists Models Summer School*. Morgan Kauffman Inc., 1988.

[18] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. Touretzky, editor, *Neural Information Processing Systems*, volume 2. Morgan Kauffman Inc., 1990.

[19] Bhaskar DasGupta and Georg Schnitger. The power of approximating: A comparison of activation functions. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Neural Information Processing Systems*, volume 5. Morgan Kauffman Inc., 1993.

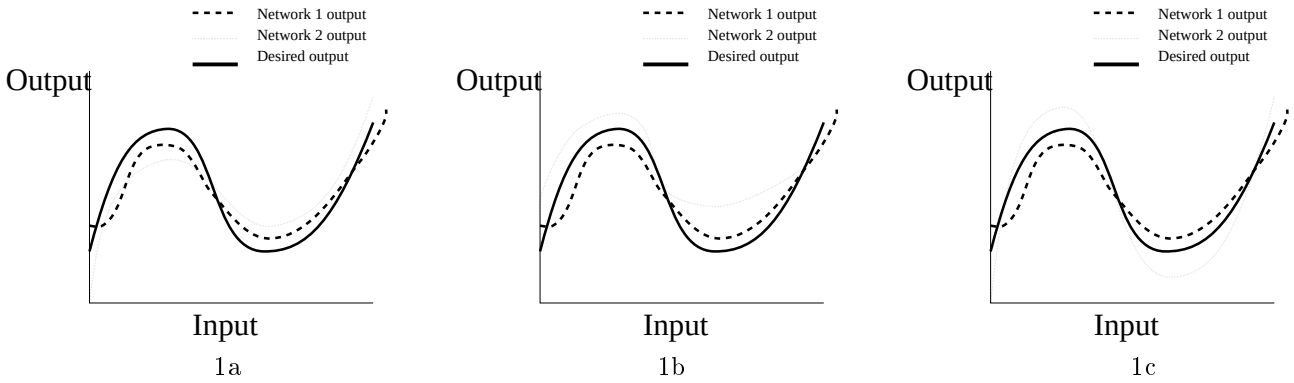

Figure 1: Correlated, Uncorrelated and Decorrelated networks. The solid curve indicates the desired output, the dashed curve indicates the first network output, and the dotted curve indicates the second network output. When two individual networks are linearly combined the correlated errors in (a) do not cancel, some correlated errors in (b) cancel but other are made worse, and the negatively correlated errors in (c) cancel.

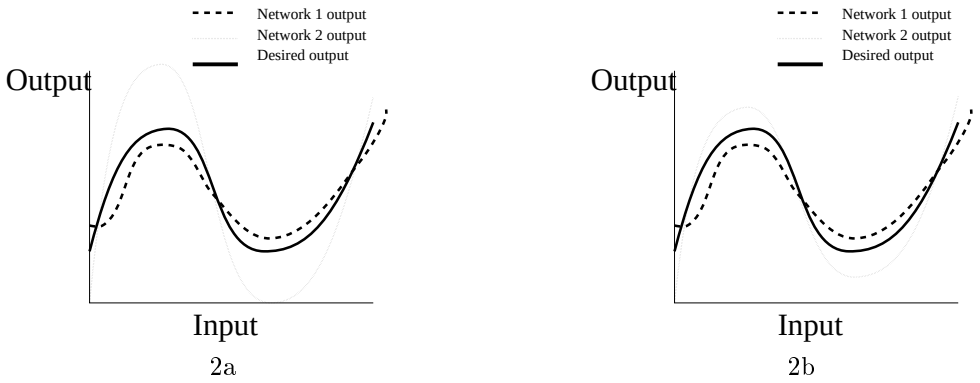

Figure 2: Two decorrelated networks before (a) and after (b) annealing the decorrelation penalty. The solid curve indicates the desired output. The dashed curve indicates the first network output and the dotted curve indicates the second, decorrelated network output.

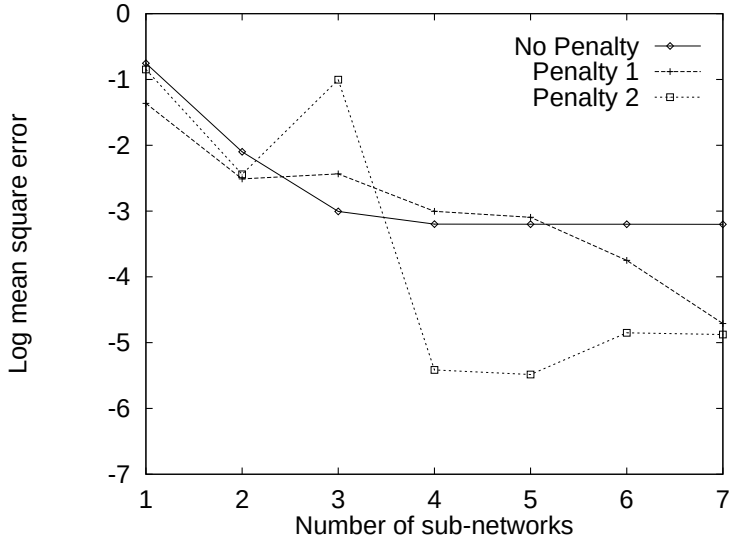

Figure 3: Performance of linearly combined networks on the "3 Parity" problem.

Network	No Penalty (μ/σ)	Penalty (μ/σ)
network 1	0.0088/0.022	0.0088/0.022
network 2	0.0079/0.019	0.0103/0.025
ensemble-net 1-2	0.0034/0.009	0.0030/0.002

Table 1: Performances of networks on the noisy sine function.

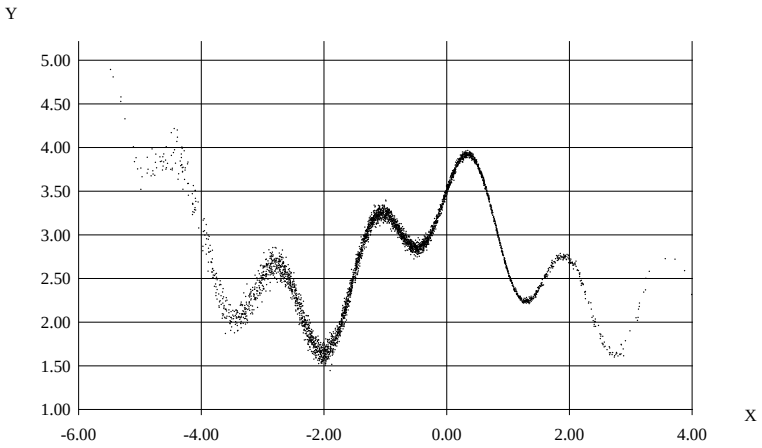

Figure 4: The noisy nonlinear nonuniformly distributed pattern data.

Network	No Penalty (μ/σ)	Penalty (μ/σ)
network 1	0.33238/0.02214	0.33238/0.02214
network 2	0.17070/0.05983	0.14912/0.00189
network 3	0.15207/0.00242	0.15506/0.00904
network 4	0.15900/0.00608	0.16635/0.02162
network 5	0.15712/0.00418	0.15882/0.00485
ensemble-net 1-2	0.15069/0.00380	0.14902/0.00126
ensemble-net 1-3	0.14874/0.00144	0.14674/0.00153
ensemble-net 1-4	0.14805/0.00218	0.14682/0.00155
ensemble-net 1-5	0.14780/0.00183	0.14641/0.00127

Table 2: Performances of networks on the noisy nonlinear test data.