Improving Regression Estimation: Averaging Methods for Variance Reduction with Extensions to General Convex Measure Optimization

by
Michael Peter Perrone
B. S., Worcester Polytechnic Institute, 1987
Sc. M., Brown University, 1989

Thesis
Submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in the Department of Physics at Brown University.

May, 1993

This dissertation by Michael Peter Perrone
is accepted in its present form by the Department of Physics
as satisfying the dissertation requirements for the degree of
Doctor of Philosophy.

Date ______________ ________________________________

Prof. Leon N Cooper

Recommended to the Graduate Council

Date ______________ ________________________________

Prof. Nathan Intrator

Date ______________ ________________________________

Prof. Charles Elbaum

Approved by the Graduate Council

Date ______________ ________________________________

Vita

Born June 5, 1965, Worcester, Massachusetts

Education

B.S., Worcester Polytechnic Institute, 1987
Sc.M., Brown University, 1989

Publications

Perrone, M. P. and L. N Cooper (1993) When Networks Disagree: Ensemble Method for Neural Networks. *Neural Networks for Speech and Image processing* ed. R. J. Mammone. [To appear].

Perrone, M. P. and L. N Cooper (1993) Coulomb Potential Learning. *In The Handbook of Brain Theory and Neural Networks* [To appear].

Perrone, M. P. and L. N Cooper (1993) Learning from what's been learned: Supervised learning in multi-neural network systems. *Proceedings of the World Conference on Neural Networks* [To appear].

Perrone, M. P. (1992) A Soft-Competitive Splitting Rule for Adaptive Tree-Structured Neural Networks. *Proceedings of the International Joint Conference on Neural Networks* pp. IV:689-693.

Perrone, M. P. and N. Intrator (1992) Unsupervised Splitting Rules for Neural Tree Classifiers. *Proceedings of the International Joint Conference on Neural Networks* pp. III:820-825.

Perrone, M. P., (1991) A Novel Recursive Partitioning Criterion. *Proceedings of the International Joint Conference on Neural Networks* p. II:989.

Acknowledgements

I am indebted to my advisor, Professor Leon N Cooper, for sharing his insights, intuitive ability for cutting straight to the heart of things, and his earnest curiousity for all things. Leon, you have made it all worthwhile.

I would also like to express my deep appreciation to Dr. Nathan Intrator who was the source of uncounted stimulating discussions including the one which lead to this dissertation. Nathan, thanks for your focus and encouragement!

The preprocessed NIST OCR data used in this thesis was provided by Nestor Inc., the Sunspot data was supplied by Andreas Weigend, and the preprocessed Face data was provided by Daniel Reisfeld. The availability of these datasets dramatically simplified my work and I am grateful.

Many thanks to Professor Charles Elbaum for a thorough reading of my thesis and a special thanks to Norma Caccia who made the lab run like clockwork and who taught me so many things about life.

To the students past and present with whom I have shared the lab, Chip Bachmann, Eugene Clothiaux, Brad Seebach, Charlie Law, Harel Shouval, Yong Liu, Luba Benuskova and Tomoko Ozeki, thanks for many useful discussions and lots of fun times. To my friends at Brown: John Moon thanks for your "Tunafish" and wild owl stories, Grep Lopinski thanks for many philosophical discussions, Scott Porter thanks for sharing some wild times, Santiago Garcia thanks for wonderful cooking, Xiaoyu Hong thanks for singing to the cows, Gerry Guralnik thanks for a great smoked bluefish mousse and Robert Beyer thanks for sharing your stories from yesteryear. Your friendships are all invaluable to me.

My deepest appreciation goes to Carmen whose warmth, sunny smile and good food got me through my dissertation - Thanks! - And to my family whose lifelong support made this all possible.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The goal of this dissertation is to present a general theoretical framework for combining populations of parametric and/or non-parametric regression estimators for classification, density estimation and regression (Perrone and Cooper, 1993c; Perrone and Cooper, 1993b). This framework has two major bulwarks.

The first is the General Ensemble Method (GEM) which can be stated as follows: Define the Generalized Ensemble Method estimator, $f_{\mathrm{GEM}}(x)$, as

$$f_{\mathrm{GEM}}(x) \equiv \sum_{i=1}^{N} \alpha_i f_i(x),$$

where the $f_i(x)$ correspond to various regression estimates, the α_i's are real and satisfy the constraint that $\sum \alpha_i = 1$. We show that the optimal α_i's are given by

$$\alpha_i = \frac{\sum_j C_{ij}^{-1}}{\sum_k \sum_j C_{kj}^{-1}}$$

where C_{ij} is the correlation between the ith and jth regression estimates. The GEM estimator is the optimal linear combination of a given set of regressors. A variation of the GEM estimator is the Basic Ensemble Estimator (BEM) which is a simple average of the $f_i(x)$'s.

The second bulwark is a series of extensions of this method to a wide variety of optimization techniques including Mean Square Error, L_p Norm Minimization, Maximum Entropy, Maximum Mutual Information, Maximum Likelihood Estimation and the Kullback-Leibler Information (or Cross Entropy).

This chapter attempts to place this theoretical framework on a firm statistical foundation by demonstrating how averaging relates to Bayesian Inference, Cross-Validation, Smoothing and Monte Carlo Approximation. In addition, this chapter draws numerous parallels to existing work on combining multiple regression estimates.

1.1 Cross-Validation and The Leave-One-Out Estimator

A regression estimate found by optimizing an empirical cost function will be biased towards the data set used. This bias, also known as over-fitting, can lead to deceptively good performance on the data. When the estimator is then tested on new data, its performance is often much worse. Cross-validation (CV) methods attempt to avoid this phenomenon by generating a performance measure on an independent test set.(Stone, 1974; Stone, 1977b; Stone, 1977a; Efron, 1979; Morgan and Bourlard, 1990a) For example, one application of cross-validation is to split the data into two pieces and use one set to converge to an empirical optimal and use the other set to decide when to terminate the convergence process. This method attempts to estimate the onset of over-fitting by "cross-validating" on an independent data set.

If a data set is prone to over-fitting, then making it smaller can only make the problem worse; thus using half the data for optimization and the other half for cross-validation, one might expect that the over-fitting problems should worsen. Ideally, we would like to use as much data as possible to construct our estimate while at the same time get the most reliable empirical estimate of the cost function for our estimate. The most common form of cross-validation, "Leave-one-out" cross-validation,(Härdle, 1990) provides a solution to this problem in the context of optimal smoothing parameter selection.

Consider the following two examples. Define the Nadaraya-Watson kernel regression function (Nadaraya, 1964) as

$$\hat{f}_\lambda(x) = \frac{\sum_{i=1}^n K_\lambda(x - x_i) y_i}{\sum_{i=1}^n K_\lambda(x - x_i)},$$

where $K_\lambda(x)$ is a kernel with width λ. The accuracy of this regression estimate varies with the choice of width. Over-fitting occurs for this regressor when λ is chosen too small. If λ is zero the kernels become delta functions and the regression function reproduces the data exactly. However if λ is chosen too large, the data becomes washed-out; and in the infinite limit, the regressor ignores the data completely. Define the mth order Smoothing Spline function (Wahba, 1990) as the minimizer of

$$\frac{1}{n}\sum_{i=1}^n (f(x_i) - \hat{f}_\lambda(x_i)))^2 + \lambda \int (\hat{f}_\lambda^{(m)}(x))^2 dx,$$

where $f(x)$ is the function which generated the data and λ is the smoothing parameter. As λ approaches 0, $\hat{f}_\lambda(x)$ overfits the data and as λ approaches infinity, $\hat{f}_\lambda(x)$ approaches the polynomial of degree $m-1$ which best fits the data.[1] In both of these smoothing examples, the very mechanism which is designed to avoid over-fitting is prone to over-fitting itself.

We can use cross-validation to choose the smoothing parameter; but instead of splitting the data into two equal sets, leave-one-out cross-validation removes a single data point, x_k, and constructs a regression estimate, $f_\lambda^{[k]}(x)$ with the remaining points. This process is repeated for each data point generating a total of n regressors. The "Ordinary Cross-Validation" function, $V_0(\lambda)$, is defined as the sum over k of the prediction error of the kth estimator on its corresponding deleted point (Wahba, 1990),

$$V_0(\lambda) = \frac{1}{n}\sum_{k=1}^n \left(y_k - f_\lambda^{[k]}(x_k) \right)^2.$$

$V_0(\lambda)$ is now minimized to find the optimal λ. Once the optimal value, λ_{opt}, is found, $\hat{f}_{\lambda_{\text{opt}}}(x)$ can then be found using all the data.

We can also use the n regression estimates generated by the leave-one-out method to estimate the integrated square error (ISE) between our regression estimate, $\hat{f}_\lambda(x)$, and the true regression function, $f(x)$. The ISE is given by

$$\text{ISE} = \int (f - \hat{f}_\lambda)^2 dx$$

[1] In general, if the smoothing term is of the form $\int (\mathcal{D}\hat{f}_\lambda(x))^2 dx$ where $\mathcal{D}$ is some operator, then $\hat{f}_\lambda(x)$ approaches the function in the null space of $\mathcal{D}$ which best fits the data.

and can be written as

$$\text{ISE} = \int f^2(x)dx - 2\int \hat{f}_\lambda(x) f dx + \int \hat{f}^2(x)dx.$$

In the case that $f(x)$ is a probability density, the second term in this equation is just twice the expected value of $\hat{f}_\lambda(x)$ and can be approximated using the leave-one-out estimates to give

$$\text{ISE} \approx \int f^2(x)dx - \frac{2}{n}\sum_{k=1}^{n} \hat{f}_\lambda^{[k]}(x_k) + \int \hat{f}^2(x)dx.$$

Note that the first term on the righthand side is a constant and can be ignored. The other terms can be calculated exactly for kernel regression and linear regression. The second term on the righthand side is known as the leave-one-out estimate of the expected value of $\hat{f}_\lambda(x)$ (Härdle, 1990),[2]

$$\widehat{E_x[\hat{f}_\lambda(x)]} = \frac{1}{n}\sum_{k=1}^{n} \hat{f}_\lambda^{[k]}(x_k).$$

This term is of interest to us since there is a striking similarity between this estimate and the BEM estimator from Section 2.2. Notice that the leave-one-out estimate is not a function of x; it is only a constant depending on the data. However, the cross-validation process which generated the leave-one-out estimators suggests that the leave-one-out heuristic and its generalization, n-fold cross-validation (leave-n-out), can be used in generating the BEM estimator.

1.2 Bayesian Inference and Monte Carlo Averaging

The cross-validation discussed in Section 1.1 can be interpreted within the much broader framework of Bayesian inference. Bayes' Theorem tells us that

$$P(w|\mathcal{D}) = \frac{P(\mathcal{D}|w)P(w)}{P(\mathcal{D})},$$

where w is a point in parameter space corresponding to a particular choice of regression function; and $\mathcal{D}$ is the observed data. In this context, $P(w|\mathcal{D})$ is the *Posterior* distribution, $P(\mathcal{D}|w)$ is the *Likelihood*, $P(w)$ is the *Prior* and $P(\mathcal{D})$ is the *Evidence* (MacKay, 1992). An optimal regression function can now be chosen by maximizing the Posterior. Note that for a fixed data set, the Evidence is a constant and can be ignored. Under the assumption of Gaussian noise, the Likelihood is (See Appendix A)

$$P(\mathcal{D}|w) \propto e^{-\text{MSE}[f]}.$$

The final step is to choose a prior for the parameters w. Under the assumption of Gaussian noise, the choice of Prior is equivalent to the choice of smoothing term in cross-validation.

[2] Efron and Stein have used a similar leave-one-out estimate to bound the variance of an estimate (Efron and Stein, 1981). The Efron-Stein Inequality states that

$$\text{VAR}[S(x_1,\ldots,x_{n-1}] \le E[\sum_{i=1}^{n}(S_i - \overline{S})^2],$$

where S_i is the ith leave-one-out estimate and $\overline{S}$ is the average of the leave-one-out estimates.

The Posterior can be used to generate the Bayes optimal regression function (See Appendix A):

$$y = \int f(x; w) P(w|\mathcal{D}) dw.$$

In general, this integral is difficult to evaluate; however since this integral has the general form of an expectation of f over parameter space, we can approximate it by

$$E_w[f(x; w)] \approx \frac{1}{n} \sum_{i=1}^{n} f(x; w_i),$$

where the w_i are independently sampled from the Posterior distribution.[3] The Strong Law of Large Numbers (Wilks, 1962) tells us that this approximation converges *almost certainly* to the true integral as n approaches infinity. This method of approximation is known as the Monte Carlo Method (Hammersley and Handscomb, 1964; Kalos and Whitlock, 1986; Mikhailov, 1992).

Monte Carlo selection of w_i has been successfully implemented for neural networks by Neal (1992, 1991). using a variation of the Metropolis Algorithm (Metropolis et al., 1953). Neal used a Hamiltonian posterior,

$$P(q, p) \propto \int \exp(-H(q, p)) dp,$$

where q is a point in parameter space, p is the associated canonical momentum, $H(q, p) = E(q) + \frac{1}{2}p^2$, and $E(q)$ is the energy. These variables were updated according to a discrete volume-preserving version of the Hamiltonian equations combined with a stochastic update step which was used to bring the system to equilibrium. Values of q were then sampled once equilibrium was achieved.

Note that Monte Carlo approximation is a special case of the BEM estimator presented in Section 2.2. The difference lies in the fact the BEM estimator places no restrictions on the process used to generate the w_i. Although relaxing this restriction abandons the guarantee of estimating the optimal Bayes regressor, it has several benefits. The BEM estimator avoids the computational overhead of stochastic sampling including achieving equilibrium and assuring that the samples drawn from the distribution are independent. Moreover, if the true prior or energy function to be minimized are not known[4], then the posterior used can only be approximated. In this case, the additional computational overhead of Monte Carlo may not improve the estimate. The BEM approach is to use a non-informative prior on the priors and average over regression estimates generated from multiple priors and energy minimization processes and is related to the Bayesian version of additive models presented by Hastie and Tibshirani (1990).

1.3 Related Algorithms

This section describes several existing methods which are related to the ensemble methods presented in this thesis. In some cases the relation is quite close but none of these methods attempts to explain why averaging reduces the MSE error in regression estimation. In addition, these algorithms have not been extended to other cost functions. These issues are addressed in Chapter 3.

[3] This process is directly analogous to the convergence of the empirical mean to the distribution mean.

[4] In general, the smoothing cost, or correspondingly the prior, is chosen heuristically. Typically, Gaussian noise is a reasonable assumption for regression problems and leads to a MSE energy function (See Appendix A) but for classification problems, a Gaussian noise assumption is obviously wrong.

1.3.1 The HARP Algorithm

The Hierarchical Adaptive Random Partitioning (HARP) Algorithm (Banan and Hjelmstad, 1992) provides and extremely simple algorithm which clearly demonstrates a fundamental aspect of averaging.

The HARP algorithm proceeds as follows:

- Randomly partition the input space.
- For each subspace, fit an approximation to the enclosed data.
- If the fit is not acceptable, partition the subspace and repeat on the children subspaces.

This algorithm rapidly generates an efficient tree structure, distributing resources where needed. One striking thing about this algorithm is that the partitioning is random - ignoring the data completely. It would seem that there must be a better way to select the partitions to improve performance. However, instead of devoting effort into optimizing the partitioning rule as has been the traditional path (Breiman et al., 1984), Banan and Hjelmstad (1992) choose to average over many randomly generated HARP estimators. They find that this averaging dramatically improves results on a toy problem.

Intuitively, what is happening is that each random partitioning of the set is generating a different, noisy, non-parametric approximation to the data. The averaging process is smoothing over the noise. If we allow averaging over all possible partitions and choose our fitting function to be a constant, we see that the averaged HARP estimator is equivalent to a kernel estimator.

1.3.2 Bayesian Pooled Estimate

Buntine and Weigend (1991) approach averaging from a Bayesian perspective to calculate a Monte Carlo approximation to the optimal regressor. Assuming that the members of a population of estimators are near local minima, they approximate the posterior distribution about each local minimum as a Gaussian,

$$P(w|\mathcal{D}, \text{near } \hat{w}) = \frac{\det(I(\hat{w}))^{\frac{1}{2}}}{(2\pi)^{\frac{1}{2}|w|}} \exp\left(-\frac{1}{2}(w-\hat{w})^T I(\hat{w})(w-\hat{w})\right) dw$$

where in the case of regression with σ^2 variance noise, $I(w)$ can be approximated by

$$I(w) \approx \frac{1}{N\sigma^2} \sum_i \frac{\mathrm{d}f(w,x_i)}{\mathrm{d}w} \frac{\mathrm{d}f(w,x_i)}{\mathrm{d}w}^T;$$

$f(w, x_i)$ is the regression function's value for data point x_i; and N is the number of data points. Within this approximation, the full posterior can be written as

$$P(w|\mathcal{D}) \approx \sum_{\hat{w}} p(\hat{w}|\mathcal{D}) P(w|\mathcal{D}, \text{near } \hat{w})$$

where $\mathcal{D}$ is the data, $\hat{w}$ is one of the local minima, and $p(\hat{w}|\mathcal{D})$ is just the proportion of nets that converged to a particular local minimum. With this posterior, the expected value of the optimal Bayes regressor (Buntine and Weigend call this the "pooled estimate") is given by

$$f_{\text{opt}}(x) = \sum_{\hat{w}} p(\hat{w}|\mathcal{D}) \left[f(x,\hat{w}) + \frac{1}{2}\text{tr}\left(I(\hat{w})^{-1} \left.\frac{\mathrm{d}^2 f(x,w)}{\mathrm{d}w^2}\right|_{w=\hat{w}} \right) \right].$$

This formulation is useful in studying the asymptotic behavior of the Bayesian optimal estimator; however since the primary premise — that the estimators are near local minima — assumes that estimators are in the asymptotic limit, it is not clear that this formulation will perform well in practice where it is rarely the case that there is enough data to assume the model is in the asymptotic regime.

1.3.3 Bayesian Belief Criterion

We shall see in Section 4.6 that one of the best methods for optical character recognition is described by Xu et al. (1992) using the framework of Pearl (1988). This method defines a confusion matrix, $M^{(k)}$, such that $M_{ij}^{(k)}$ is the number of patterns from class i assigned a label of class j by the kth classifier. The probability of a pattern belonging to class i when classifier k says it belongs to class j is given by

$$P(x \in C_i | e_k(x) = j) = \frac{M_{ij}^{(k)}}{\sum_i M_{ij}^{(k)}}$$

where $e_k(x)$ is the classification from the kth classifier for pattern x. Assuming that these probabilities are all independent, the "belief" in the ith class for pattern x is

$$B(i) = \eta \prod_{k=1}^{K} P(x \in C_i | e_k(x) = j_k)$$

where η is a normalization constant, K is the number of classifiers and j_k is the classification from classifier k for pattern x.[5] Xu then defines a Belief Criterion[6]

$$E(x) = \begin{cases} j, & \text{if} B(j) = \max_i B(i) \geq \alpha \\ \text{reject}, & \text{Otherwise} \end{cases}$$

where α is a rejection threshold. Through varying α the ratio of the rejection rate to the misclassification rate can be optimized. This method has an advantage over the GEM method in that it does not require the inversion of a matrix (See Section 2.4); however the belief is a product over many random variables and may lead to similar instabilities due to noise in the counting statistics of the the confusion matrix. Regardless, this method seems very promising.

Xu presents several other confidence measures but only one of them is comparable in performance to the Bayesian approach above and that is an approach based on Demptser-Schafer Theory (Mandler and Schuermann, 1988; Schafer, 1976; Schafer and Logan, 1987). Since this second approach does not out-perform the Bayesian approach on our measure[7] we do not discuss it further.

1.3.4 Sieving

Drucker et al. (1992) implement a simple averaging technique and appeal to probably approximately correct (PAC) learning (Valiant, 1984) to justify their averaging (Schapire, 1990). The PAC result shows that averaging over an infinite population of independent classifiers each which has an error rate less than $\frac{1}{2}$ will result in zero error.

[5] This "belief" is analogous to the maximum likelihood confidence measure defined in Section 4.4.
[6] In Section 4.4 this is called a rejection criterion.
[7] Our criterion is FOM and is described in Section 4.3.

One interesting aspect of Drucker's work is the suggestion of a "Sieving" method. Motivated by a desire to save time in serial computation of the networks in a population, Drucker suggests passing an input pattern to a subsequent network only if the current network activity is below a threshold chosen to reject 50 percent of the patterns reaching that network.

1.3.5 Multi-Resolution Hierarchical Filtering

Scofield et al. (1991) use a hierarchical model for combining networks which adaptively trains according to the level of difficult in classification. A set of networks is constructed with networks of varying priority. Networks with high priority have low feature resolution and networks with low priority have high resolution (Reilly et al., 1987; Moody, 1989; Perrone and Cooper, 1993a). When a pattern is presented to the ensemble of networks, the high priority networks are processed first. The classification of lower priority networks is only considered when the higher priority networks fail or are below some threshold.

This method is similar to Drucker's Sieving except that the multi-resolution hierarchy has the advantage of being able to train using the hierarchy whereas Drucker's Sieving only comes after training.

Another method for hierarchical filtering was presented by Perrone (1992a, 1992b, 1991) in which filtering was performed in a tree structure neural network (Breiman et al., 1984; Sankar and Mammone, 1991). This method in effect replaces resolution with complexity and constructs a multi-complexity hierarchy.

1.3.6 Synergy

Lincoln and Skrzypek (1990) suggest using both a simple average

$$\overline{f}(x) = \sum_i f_i(x)$$

and a weighted average

$$\overline{f}(x) = \sum_i w_i f_i(x).$$

For the weighted average, they propose the following learning rule

$$w_i = \eta w_i \frac{e}{e_i}$$

where e_i is the distance of the ith estimate from the true value, e is the average of the e_i's, and η is a fixed stepsize. They do not say from where this learning rule comes and it is not clear that it is a desirable learning rule since its only fixed points occur when all the weights are zero or when all the e_i are equal. This fact can be see by averaging over all the data.

We propose the following iterative update rule

$$\Delta w_i = -\eta(\overline{f}(x) - f(x))f_i(x)$$

which is equivalent to stochastic optimization of the MSE of the $\overline{f}(x)$ estimator when $f(x)$ is the function to which we are regressing. In other words, one could use the perceptron learning algorithm (Rumelhart et al., 1986) using the output of the regressors as input. (See Section 2.4 for a detailed discussion.)

1.3.7 Averaging and the Bias/Variance Dilemma

Geman et al. (1992) use simple averaging to analyze the Bias/Variance Dilemma. The MSE of an estimator can be written as the sum of two term: the bias and the variance. Geman proposes to approximate these two terms using

$$\text{Bias}[\hat{f}(x)] = \left(\overline{f}(x) - E[y|x]\right)^2$$

and

$$\text{Variance}[\hat{f}(x)] = \frac{1}{n}\sum_{i=1}^{N}\left(\hat{f}_i(x) - E[y|x]\right)^2$$

where N is the number of estimates in the population and $\hat{f}(x) = \frac{1}{n}\sum_{i=1}^{N}\hat{f}_i(x)$. Using these approximations, we can also estimate the integrated bias and the integrated variance by averaging over all of the data. Geman uses these estimates to clearly show how bias and variance depend on complexity. The fundamental trade-off is that as complexity increases, bias decreases and variance increases; as complexity decreases, variance decreases and bias increases.

Geman's analysis suggests a very important result: In Section 3.7, we show that averaging is a variance reduction technique. In fact, using averaging it is possible to make the variance of an estimator arbitrarily small. Thus if we use models with very high complexity we can reduce the bias and if we average these models we can reduce the variance. Both, in principle can be reduced arbitrarily! Unfortunately there is a catch: The values we are minimizing are estimates from the data not from the true data distribution. Thus, there will be a point when the complexity of the model goes beyond what is justified by the data and the estimates will no longer be accurate. We must either have a method for stopping the increase in complexity (e.g. the standard approach is cross-validation) or we must increase the data set fast enough as the complexity increases.

1.3.8 Hansen's Ensemble Performance Estimate

Hansen et al. (1992,1990) develop a theoretical result for a special case of averaging which they call a *plurality decision*. This is a "majority-rule" where each estimator has an equal vote during classification.[8] Starting with a population of N classifiers voting for M classes, Hansen assumes that all of the classifiers errors are independent[9] and proves that the average ensemble performance, P, is given by

$$P = \int P(\theta)\mu(\theta)d\theta$$

where θ is the fraction of incorrect classifiers, $\mu(\theta)$ is the probability of θ, and $P(\theta)$ is given by

$$P(\theta) = \sum_{n=0}^{1+\frac{N-1}{M}} \binom{N}{n} \theta^n(1-\theta)^{N-n} + \sum_{1+\frac{N-1}{M}}^{N} \binom{N}{n} \theta^n(1-\theta)^{N-n} H(n)$$

and $H(n)$ is given by

$$H(n) = 1 - \frac{\sum_{k=0}^{M-1}(-1)^k \binom{M-1}{k} \binom{N-n(k+1)+M-2}{N-n(k+1)}}{\binom{N-n+M-2}{M-2}}.$$

[8] Obviously this method can only be used for classification - not regression.

[9] Unfortunately, this assumption will rarely be true in practice.

We use the convention that any binomial coefficient with negative entries are zero. This expression is exact but is not useful until we have knowledge of $\mu(\theta)$ which in general is not known. Hansen suggests that an entropic prior is appropriate but in the end claims that the real measure of ensemble performance must come from cross-validation.

1.3.9 Discussion

The algorithms above are all special cases of a general approach outlined by Wolpert (1990) in the case of neural network training (called *stacked generalization*), and by Breiman (1992) (Breiman, 1992) in the context of regression (called it stacked regression). The basic idea behind these approaches is to use one regression function to "learn" from the "mistakes" of other regression functions. For example, in the context of classification, a neural network could be trained to identify which of a set of previously trained neural networks is correct in various regions of input space and then by combining them accordingly, improve the overall classification performance. Repeating this process leads to the image of "stacking" the estimators on top of one another.

Variations of stacking have been used by many researchers (Cooper, 1991; Reilly et al., 1988; Reilly et al., 1987; Scofield et al., 1991; Jacobs et al., 1991; Pearlmutter and Rosenfeld, 1991; Xu et al., 1990). However, like Wolpert and the algorithms presented in the preceding sections, few of these algorithms offered more than heuristic explanations for why performance should improved.

In this thesis, we move towards a more theoretical understanding of how "stacking" improves performance. We will see how this theoretical approach points to the importance of the convexity property in the case of linear combinations of regression estimates. Realization of the importance of the convexity property will then lead us to a broad generalization of the averaging methods presented.

Chapter 2

Ensemble Methods for Improving Regression Estimates

2.1 Introduction

Hybrid or multi-neural network systems have been frequently employed to improve results in classification and regression problems (Cooper, 1991; Reilly et al., 1988; Reilly et al., 1987; Scofield et al., 1991; Baxt, 1992; Bridle and Cox, 1991; Buntine and Weigend, 1992; Hansen and Salamon, 1990b; Intrator, 1993; Jacobs et al., 1991; Lincoln and Skrzypek, 1990; Neal, 1993; Neal, 1992; Pearlmutter and Rosenfeld, 1991; Wolpert, 1990; Xu et al., 1992; Xu et al., 1990). Among the key issues are how to design the architecture of the networks; how the results of the various networks should be combined to give the best estimate of the optimal result; and how to make best use of a limited data set. In what follows, we address the issues of optimal combination and efficient data usage in the framework of ensemble averaging.

In this chapter we are concerned with using the information contained in a set of regression estimates of a function to construct a better estimate. The statistical resampling techniques of jackknifing, bootstrapping and cross-validation have proven useful for generating improved regression estimates through bias reduction (Efron, 1982; Miller, 1974; Stone, 1974; Gray and Schucany, 1972; Härdle, 1990; Wahba, 1990, for review). We show that these ideas can be fruitfully extended to general regression problem and in particular to neural networks by using the ensemble methods presented in this chapter. In addition to the bias reduction properties of the re-sampling techniques, we will show in Chapter 3 that ensemble methods are performing variance reduction. The basic idea behind these resampling techniques is to improve one's estimate of a given statistic, $\hat{\theta}$, by combining multiple estimates of θ generated by subsampling or resampling of a finite data set. The jackknife method (Efron, 1982; Härdle, 1990) involves removing a single data point from a data set, constructing an estimate of θ with the remaining data, testing the estimate on the removed data point and repeating for every data point in the set. One can then, for example, generate an estimate of $\hat{\theta}$'s variance using the results from the estimate on all of the removed data points. This method has been generalized to include removing subsets of points. The bootstrap method (Lepage and Billard, 1992; Hall, 1992; Carroll and Ruppert, 1988a) involves generating new data sets from one original data set by sampling randomly with replacement. These new data sets can then be used to generate multiple estimates for θ. In cross-validation (Stone, 1974), the original data is divided into two sets: one which is used to generate the estimate of θ and the other which is used to test this estimate. Cross-Validation is widely used in neural network training to avoid over-fitting

(Morgan and Bourlard, 1990b; Baldi and Chauvin, 1991; Moore, 1992; Moody and Utans, 1992; Koistinen and Holmstrom, 1992; Montana, 1992; Liu, 1993; Finnoff et al., 1993). However, the jackknife and bootstrap methods are not commonly used in neural network training for two reasons. First, the re-sampling schemes incur a large computational overhead from the need to re-calculate the regression estimate for each sub-sample of the data and therefore can slow computation by a factor of order N where N is the number of data samples. Second, and perhaps more important, is the fact that in non-parametric regression problems it is not clear how to use the multiplicity of regression estimates generated by re-sampling. For parametric models, one can average in parameter space over the population of regression estimates generated which may be a reasonable approach when the parameters have some specific significance whether it be physical or otherwise such as the decay constant for some process evolving in time. However, in non-parametric problems where the individual parameters have little or no significance independent of all of the other parameters in the model, averaging in parameter space is not justified. We will show how we can overcome this second problem and apply the re-sampling techniques to non-parametric regression.

Resampling techniques can be used to generate multiple distinct networks from a single training set. For example, resampling in neural net training frequently takes the form of repeated on-line stochastic gradient descent of randomly initialized nets. However, unlike the combination process in parametric estimation which usually takes the form of a simple average in parameter space, the parameters in a neural network take the form of neuronal weights which generally have many different local minima. Therefore we can not simply average the weights of a population of neural networks and expect to improve network performance. Because of this fact, one typically generates a large population of resampled nets and chooses the one with the best performance and discards the rest. This process is very inefficient. Below, we present ensemble methods which avoid this inefficiency and avoid the local minima problem by averaging in functional space not parameter space. In addition we show that the ensemble methods actually benefit from the existence of local minima and that within the ensemble framework, the statistical resampling techniques have very natural extensions. All of these aspects combined provide a general theoretical framework for network averaging which in practice generates significant improvement on real-world problems.

There is a very strong connection between the methods described in this chapter and the standard Monte Carlo integration techniques (Kalos and Whitlock, 1986; Mikhailov, 1992). Therefore it is possible to apply the methods described in this chapter to the standard Monte Carlo problems. The difference between these two approaches is that the Monte Carlo method assumes that one is sampling from the true probability distribution of the problem. The methods here relax this assumption since the true distribution is only empirically known through the observed data and cannot be calculated explicitly or modelled beyond what is known from the data. In addition, the methods presented here do not attempt to solve the integral problem exactly as Monte Carlo does. Here it is shown that by using more of the information available to us, one can improve our regression estimate. Neal (Neal, 1993; Neal, 1992) has shown that neural network performance can be dramatically improved by using Monte Carlo methods; however, this approach requires additional assumptions about the distribution and a significant increase in computation.

In this chapter, an optimal hybrid combination of regression estimates based on ensemble averaging and closely related to Monte Carlo integration is presented which avoids over-fitting by variance reduction smoothing, which can use the standard statistical resampling techniques to perform bias reduction, and which benefits from the existence of local minima. The chapter is organized as follows. In Section 2.2, the Basic Ensemble Method (BEM) for generating improved regression estimates from a population of estimates by averaging in functional space is described.

In Section 2.3, simple examples are given to motivate the BEM estimator. Section 2.4, describes the Generalized Ensemble Method (GEM) and prove that it produces an estimator which always reduces the mean square error. Techniques for improving the performance of the ensemble methods are described in Section 2.5. Section 2.6 contains a discussion of BEM and GEM.

2.2 Basic Ensemble Method

Consider the following regression problem

$$y = f(x) + n$$

where y is a random variable with mean $f(x) = E[y|x]$ and n is independent zero-mean noise.[1] We present the Basic Ensemble Method (BEM) which combines a population of regression estimates, $\hat{f}_i(x)$, to estimate a function $f(x)$.

Suppose that we have two finite data sets whose elements are all independent and identically distributed random variables: a training data set $\mathcal{A} = \{(x_m, y_m)\}$ and a cross-validatory data set $\mathcal{CV} = \{(x_l, y_l)\}$. Further suppose that we have used $\mathcal{A}$ to generate a set of functions, $\mathcal{F} = f_i(x)$, each element of which approximates $f(x)$.[2] We would like to find the best approximation to $f(x)$ using $\mathcal{F}$.

One common choice is to use the *naive estimator*, $f_{\text{Naive}}(x)$, which minimizes the empirical mean square error relative to $f(x)$,[3]

$$\text{MSE}[f_i] = E_{\mathcal{CV}}[(y_l - f_i(x_l))^2],$$

thus

$$f_{\text{Naive}}(x) = \arg\min_i \{\text{MSE}[f_i]\}.$$

This choice is unsatisfactory for two reasons: First, in selecting only one regression estimate from the population of regression estimates represented by $\mathcal{F}$, we are discarding potentially useful information that is stored in the discarded regression estimates; second, since the $\mathcal{CV}$ data set is random, there is a certain probability that some other network from the population will perform better than the naive estimate on some other previously unseen data set sampled from the same distribution. A more reliable estimate of the performance on previously unseen data is the average of the performances over the population $\mathcal{F}$. Below, we will see how we can avoid both of these problems by using the BEM estimator, $f_{\text{BEM}}(x)$, and thereby generate an improved regression estimate.

Define the *misfit* of function $f_i(x)$, the deviation from the true solution, as $m_i(x) \equiv f(x) - f_i(x)$. The empirical mean square error can now be written in terms of $m_i(x)$ as

$$\text{MSE}[f_i] = E[m_i^2].$$

The average mean square error is therefore

$$\overline{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N} E[m_i^2].$$

[1] The noise for minimizing the MSE is assumed to be Gaussian; but this assumption is not necessary for what follows.

[2] For our purposes, it does not matter how $\mathcal{F}$ was generated, unlike Monte Carlo. In practice we will use a set of backpropagation networks trained on the $\mathcal{A}$ data set but started with different random weight configurations. This replication procedure is standard practice when trying to optimize neural networks.

[3] Here, and in all of that follows, the expected value is taken over the cross-validatory set $\mathcal{CV}$.

Define the BEM regression function, $f_{\mathrm{BEM}}(x)$, as

$$f_{\mathrm{BEM}}(x) \equiv \frac{1}{N}\sum_{i=1}^{N} f_i(x) = f(x) - \frac{1}{N}\sum_{i=1}^{N} m_i(x)$$

If we now assume that the $m_i(x)$ are mutually independent with zero mean, [4] we can calculate the mean square error of $f_{\mathrm{BEM}}(x)$ as

$$\begin{aligned}
\mathrm{MSE}[f_{\mathrm{BEM}}] &= E\Big[(\frac{1}{N}\sum_{i=1}^{N} m_i)^2\Big] \\
&= \frac{1}{N^2}E\Big[\sum_{i=1}^{N} m_i^2\Big] + \frac{1}{N^2}E\Big[\sum_{i\neq j} m_i m_j\Big] \\
&= \frac{1}{N^2}E\Big[\sum_{i=1}^{N} m_i^2\Big] + \frac{1}{N^2}\sum_{i\neq j} E[m_i]E[m_j] \\
&= \frac{1}{N^2}E\Big[\sum_{i=1}^{N} m_i^2\Big], \qquad (2.1)
\end{aligned}$$

which implies that

$$\mathrm{MSE}[f_{\mathrm{BEM}}] = \frac{1}{N}\overline{\mathrm{MSE}}. \qquad (2.2)$$

This is a powerful result because it tells us that by averaging regression estimates, we can reduce our mean square error by a factor of N when compared to the population performance: By increasing the population size, we can in principle make the estimation error arbitrarily small! In practice however, as N gets large our assumptions on the misfits, $m_i(x)$, eventually breakdown. In particular, the assumption that $E[m_i m_j] = E[m_i]E[m_j]$ is no longer valid. This effect is examined experimentally in Section 4.7.

Consider the individual elements of the population $\mathcal{F}$. These estimators will more or less follow the true regression function. If we think of the misfits functions as random noise functions added to the true regression function and these noise functions are uncorrelated with zero mean, then the averaging of the individual estimates is like averaging over the noise. In this sense, the ensemble method is smoothing in functional space and can be thought of as a regularizer with a smoothness assumption on the true regression function. (See Section 3.8 for more on regularizers.)

An additional benefit of the ensemble method's ability to combine multiple regression estimates is that the regression estimates can come from many different sources. This fact allows for flexibility in the application of the ensemble method. For example, the regression estimates can have different functional forms; or can be selected using different optimization (i.e. "training") algorithms; or can be selected by optimizing over different data sets. This last option - optimizing on different data sets - has important ramifications. One standard method for avoiding over-fitting during training is to use a cross-validatory hold-out set.[5] The problem is that since we use cross-validation to avoid over-fitting, each regression estimate is never trained on the hold-out data (i.e. the cross-validatory data set) and therefore, each regression estimate "sees" only part of the data and may be missing valuable information about the distribution

[4] We relax these assumptions in Section 2.4 where we present the Generalized Ensemble Method.

[5] The cross-validatory hold-out set is a subset of the total data available to us and is used to determine when to stop training. The hold-out data is not used to train.

of the data particularly if the total data set is small. This will always be the case for a single regression estimate using a cross-validatory stopping rule. However, this is not a problem for the ensemble estimator. When constructing our population, $\mathcal{F}$, we can train each regression estimate on the entire training set and let the smoothing property of the ensemble process remove any over-fitting or we can train each regression estimate in the population with a different split of training and hold-out data. In this way, the population as a whole will have seen the entire data set while each regression estimate has avoided over-fitting by using a cross-validatory stopping rule. Thus the ensemble estimator will see the entire data set while the naive estimator will not. In general, with this framework we can now easily extend the statistical jackknife, bootstrap and cross-validation techniques (Efron, 1982; Miller, 1974; Stone, 1974) to find better regression functions.

2.3 Intuitive Illustrations

In this section, we try to motivate the averaging method presented in Section 2.2 with two toy examples which illustrate the averaging principle which is at the heart of the ensemble methods presented in this chapter.

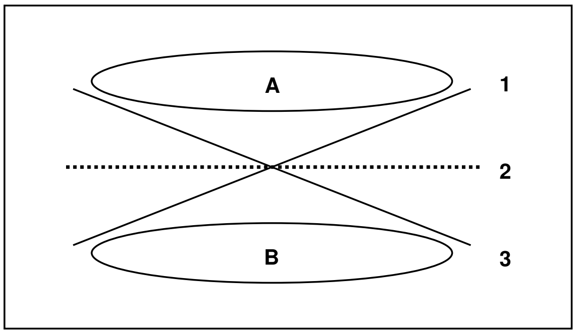


Figure 2.1: Toy classification problem. Hyperplanes 1 and 3 solve the classification problem for the training data but hyperplane 2 is the optimal solution. Hyperplane 2 is the average of hyperplanes 1 and 3.

For our first example, consider the classification problem depicted in Fig. 2.1. Regions A and B represent the training data for two distinct classes which are Gaussianly distributed. If we train a perceptron on this data, we find that hyperplanes 1, 2 and 3 all give perfect classification performance for the training data; however only hyperplane 2 will give optimal generalization performance. Thus, if we had to choose a naive estimator from this population of three perceptrons, we would be more likely than not to choose a hyperplane with poor generalization performance. For this problem, it is clear that the BEM estimator (i.e. averaging over the 3 hyperplanes) is more reliable. In very high dimensional spaces such as those used in pattern recognition problems, the problem depicted in Fig. 2.1 is likely due to the inherent sparsity of the data (Duda and Hart, 1973).

Consider a second example. Approximate a Gaussian distribution given two estimates shown in Fig. 2.3. If we must choose either one or the other of these estimates we will incur a certain mean square error; however, if we average these two functional estimates the mean square error is dramatically reduced. In Fig. 2.3, the ensemble average of the two estimates from Fig. 2.3 is presented. Comparing Fig. 2.3 to Fig. 2.3, it is clear that the ensemble estimate is much better than either of the individual estimates. Figure 2.3 compares the square error of each of the estimates.

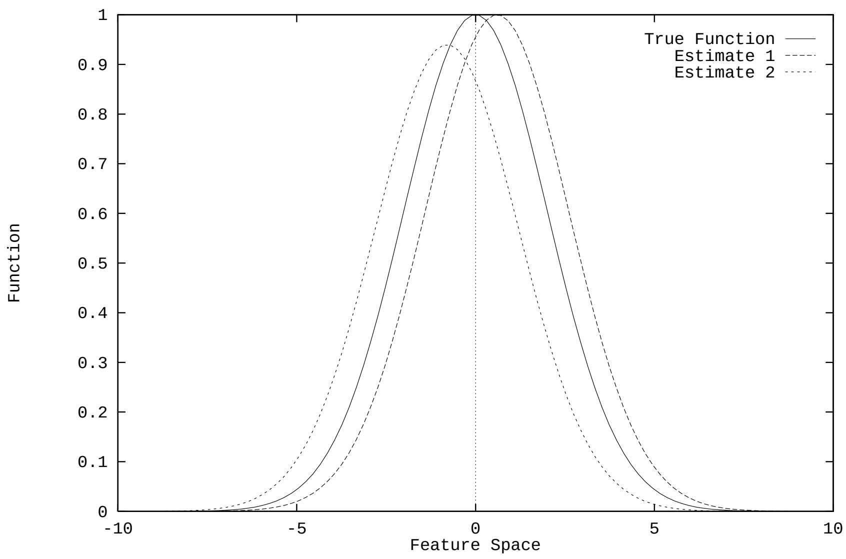


Figure 2.2: Two randomly chosen Gaussian estimates compared to the true Gaussian

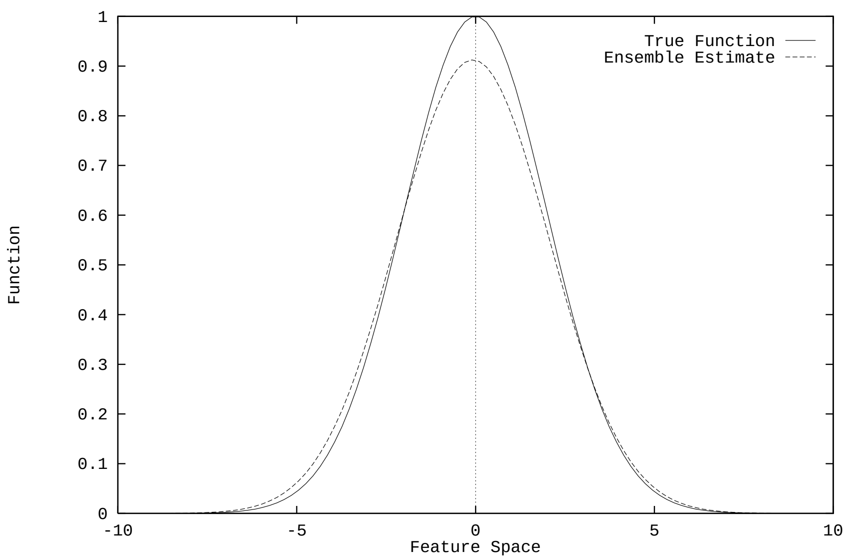


Figure 2.3: Ensemble average estimate compared to the true Gaussian

It is instructive to push this simple example further. Suppose that $x \sim \mathcal{N}(0, \sigma^2)$ and we are

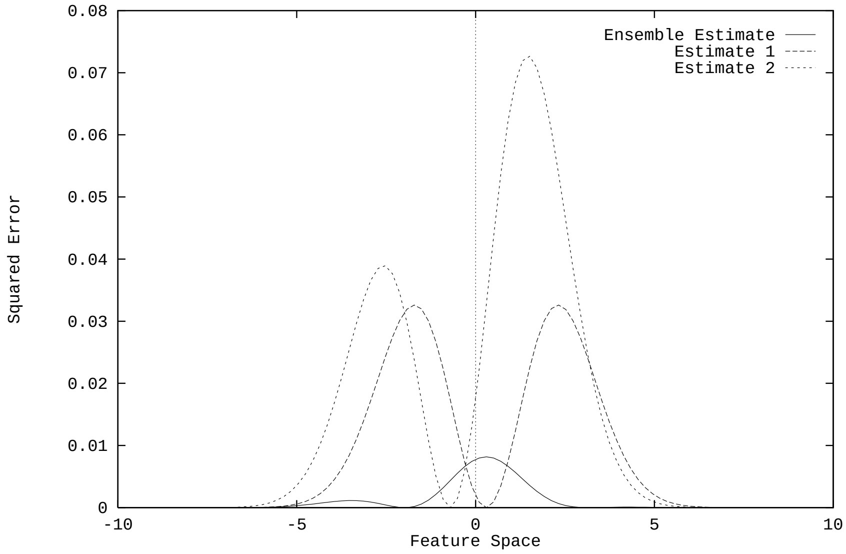


Figure 2.4: Square error comparison of the three estimates. Notice that the ensemble estimate gives the smallest square error

given $\mathcal{D} \equiv \{x_i\}_{i=1}^{i=N}$ and σ^2. We can estimate the true Gaussian by estimating its mean with

$$\mu \equiv \frac{1}{N}\sum_{j=1}^{N} x_j$$

or we can use a modification of the Jackknife Method (Gray and Schucany, 1972) to construct a population of estimates from which we can construct an ensemble estimator. Define the ensemble estimate as

$$g_{\text{Ensemble}}(x) \equiv \frac{1}{N}\sum_{j=1}^{N} g(x;\mu_{(-i)})$$

where

$$\mu_{(-i)} \equiv \frac{1}{N-1}\sum_{j\neq i} x_j$$

and

$$g(x;\alpha) \equiv \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\alpha)^2}{\sigma^2}}.$$

We now explicitly compare these two estimates using the mean integrated square error (MISE) of the estimates,

$$\text{MISE}[g(x;\alpha)] = E_{\mathcal{D}}[\int_{-\infty}^{+\infty}(g(x;\alpha)-g(x;0))^2dx].$$

Define $\Delta(\alpha) \equiv \int_{-\infty}^{\infty} e^{-\alpha x^2}dx$ and $\gamma \equiv 1/\sqrt{2\pi\sigma^2}$, and note that

$$\int_{-\infty}^{\infty} e^{-\alpha(x-\mu)^2}e^{-\alpha x^2}dx = e^{-\alpha\mu^2/2}\Delta(2\alpha) \qquad (2.3)$$

and

$$\int_{-\infty}^{\infty} e^{-\beta x^2} e^{-\alpha x^2} dx = \Delta(\alpha)\sqrt{\frac{\alpha}{\beta+\alpha}}. \tag{2.4}$$

The integrated square error for $g(x;\mu)$ is given by

$$\mathrm{ISE}[g(x;\mu)] = \gamma^2 \int_{-\infty}^{\infty} \left(e^{-\frac{(x-\mu)^2}{\sigma^2}} - 2e^{-\frac{(x-\mu)^2}{2\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} + e^{-\frac{x^2}{\sigma^2}} \right) dx.$$

The third term of the integral is just $\Delta(\sigma^{-2})$. After a linear change of variables, the first term of the integral is also $\Delta(\sigma^{-2})$. Using Equation (2.3) on the second term in the integrand gives

$$\mathrm{ISE}[g(x;\mu)] = 2\gamma^2 \left(1 - e^{-\mu^2/4\sigma^2}\right) \Delta(\sigma^{-2}).$$

Since μ is the average of N i.i.d. zero mean Gaussian variables, it is distributed as a zero mean Gaussian variable with variance given by σ^2/N. Using this information, the expected value of the ISE of $g(x;\mu)$ is given by

$$\mathrm{MISE}[g(x;\mu)] = 2\gamma^3 \Delta(\sigma^{-2})\sqrt{N} \int_{-\infty}^{\infty} \left(1 - e^{-\mu^2/4\sigma^2}\right) e^{-N\mu^2/2\sigma^2} d\mu$$

Using Equation (2.4) and the definition for $\Delta(\alpha)$ gives

$$\mathrm{MISE}[g(x;\mu)] = \frac{2}{\sqrt{\pi\sigma^2}} \left(1 - \sqrt{\frac{2N}{2N+1}}\right).$$

Note that $\frac{n-1}{2}(\mu_i - \mu_j)$ for $i \neq j$ is a Gaussian random variable with zero mean and variance given by $\sigma^2/2$. This fact combined with Equations (2.3) and (2.4) can be used to evaluate the MISE for $g_{\mathrm{Ensemble}}(x)$. Following the same arguments as were used to evaluate the MISE for $g(x;\mu)$ gives

$$\mathrm{MISE}[g_{\mathrm{Ensemble}}(x)] = \frac{1}{\sqrt{\pi\sigma^2}} \left(1 + \frac{1}{N} + \frac{(N-1)^2}{N\sqrt{(N-1)^2+1}} - 2\sqrt{\frac{2N-2}{2N-1}}\right).$$

The $\mathrm{MISE}[g_{\mathrm{Ensemble}}(x)]$ and $\mathrm{MISE}[g(x;\mu)]$ are graphed in Figure 2.5. The figure shows that the $\mathrm{MISE}[g_{\mathrm{Ensemble}}(x)]$ is always lower than the $\mathrm{MISE}[g(x;\mu)]$ for $n > 2$. This effect is most significant for small n.

2.4 Generalized Ensemble Method

In this section we extend the results of Section 2.2 to a generalized ensemble technique that generates a regression estimate which is as low or lower than both the best individual regressor, $f_{\mathrm{Naive}}(x)$, and the basic ensemble regressor, $f_{\mathrm{BEM}}(x)$, and which avoids overfitting the data. It is the best possible of any linear combination of the elements of the population $\mathcal{F}$ based on the empirical MSE.

Define the Generalized Ensemble Method estimator, $f_{\mathrm{GEM}}(x)$, as

$$f_{\mathrm{GEM}}(x) \equiv \sum_{i=1}^{N} \alpha_i f_i(x) = f(x) + \sum_{i=1}^{N} \alpha_i m_i(x),$$

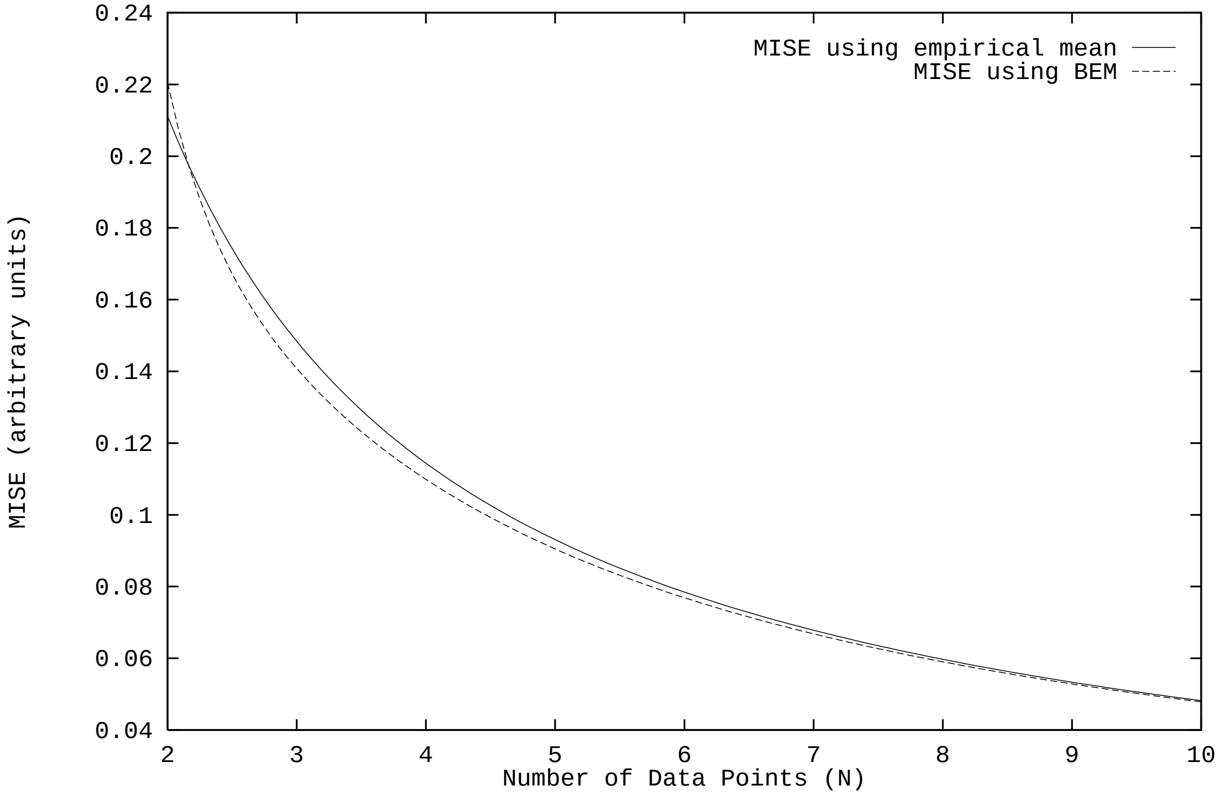


Figure 2.5: MISE of the BEM estimator compared to MISE of single estimator in arbitrary units.

where the α_i's are real and satisfy the constraint that $\sum \alpha_i = 1$. We want to choose the α_i's so as to minimize the MSE with respect to the target function $f(x)$. If again we define $m_i(x) \equiv f(x) - f_i(x)$ and in addition define the symmetric correlation matrix

$$C_{ij} \equiv E[m_i(x) m_j(x)] \tag{2.5}$$

then we find that we must minimize

$$\mathrm{MSE}[f_{\mathrm{GEM}}] = \sum_{i,j} \alpha_i \alpha_j C_{ij}. \tag{2.6}$$

We now use the method of Lagrange multipliers to solve for α_k. We want α_k such that $\forall\ k$

$$\partial_{\alpha_k} \left[\sum_{i,j} \alpha_i \alpha_j C_{ij} - 2\lambda (\sum_i \alpha_i - 1) \right] = 0.$$

This equation simplifies to the condition that

$$\sum_k \alpha_k C_{kj} = \lambda.$$

If we impose the constraint, $\sum \alpha_i = 1$, we find that

$$\alpha_i = \frac{\sum_j C_{ij}^{-1}}{\sum_k \sum_j C_{kj}^{-1}}. \tag{2.7}$$

If the $m_i(x)$'s are uncorrelated and zero mean, $C_{ij} = 0 \ \forall \ i \neq j$ and the optimal α_i's have the simple form

$$\alpha_i = \frac{\sigma_i^{-2}}{\sum_j \sigma_j^{-2}},$$

where $\sigma_i^2 \equiv C_{ii}$, which corresponds to the intuitive choice of weighting the f_i's by the inverse of their respective variances and normalizing. Combining equations (2.6) and (2.7), we find that the optimal MSE is given by

$$\mathrm{MSE}[f_{\mathrm{GEM}}] = \left(\sum_{ij} C_{ij}^{-1} \right)^{-1}. \tag{2.8}$$

Note that the process described above may introduce bias to whatever data set[6] is used to generate C_{ij}. This bias is due to the fact that the C we calculate is the sample correlation matrix not the true correlation matrix, so C is a random variable as are $\mathrm{MSE}[f_{\mathrm{GEM}}]$ and the optimal α_i's. Thus noise in the estimate of C can lead to bad estimates of the optimal α_i's. If needed, we can get a less biased estimate of C^{-1} by using a jackknife procedure (Gray and Schucany, 1972) on the data used to generate C or a biasing method such as ridge regression (Vinod and Ullah, 1981). If we can not trust our estimates of C_{ij} then we can fall back to the BEM estimator.

The results in this section depend on two assumptions: The rows and columns of C are linearly independent and we have a reliable estimate of C. In certain cases where we have nearly duplicate networks in the population $\mathcal{F}$, we will have nearly linearly dependent rows and columns in C which will make the inversion process very unstable and our estimate of C^{-1} will be unreliable. In these cases, we can use heuristic techniques to sub-sample the population $\mathcal{F}$ to assure that C has full rank (See Section 2.5). In practice, the increased stability produced by removing near degeneracies outweighs any information lost by discarding nets.

Note also that the BEM estimator and the naive estimator are both special cases of the GEM estimator and therefore $\mathrm{MSE}[f_{\mathrm{GEM}}]$ will always be less than or equal to $\mathrm{MSE}[f_{\mathrm{BEM}}]$ and $\mathrm{MSE}[f_{\mathrm{Naive}}]$. An explicit demonstration of this fact can be seen by comparing the respective MSE's under the assumption that the $m_i(x)$'s are uncorrelated and zero mean. In that case, comparing Equations (2.1) and (2.8), we have

$$\mathrm{MSE}[f_{\mathrm{BEM}}] = \frac{1}{N^2} \sum_i \sigma_i^2 \geq \left(\sum_i \sigma_i^{-2} \right)^{-1} = \mathrm{MSE}[f_{\mathrm{GEM}}],$$

with equality only when all of the σ_i are identical. This relation is easily proven using the fact that $\frac{a}{b} + \frac{b}{a} \geq 2 \ \forall \ a, b > 0$. Similarly we can write

$$\mathrm{MSE}[f_{Naive}] = \sigma_{\mathrm{min}}^2 \geq \left(\sum_i \sigma_i^{-2} \right)^{-1} = \mathrm{MSE}[f_{\mathrm{GEM}}].$$

Thus we see that the GEM estimator provides the best estimate of $f(x)$ in the mean square error sense.

[6] Typically, the cross-validation set is also used to generate our estimate of C_{ij} although we could also use the training data or some combination of the two.

2.5 Improving BEM and GEM

One simple extension of the ensemble methods presented in this chapter is to consider the BEM and GEM estimators of all of the possible populations which are subsets of the original network population $\mathcal{F}$. [7] All the information we need to perform subset selection is contained in the correlation matrix, C, which only has to be calculated once.

In principle, the GEM estimator for $\mathcal{F}$ will be a better estimator than the GEM estimator for any subset of $\mathcal{F}$ since it will always choose the best linear combination of estimators in the populations which includes the possibility of selecting all the nets with equal weights or of selecting only one regression function from the population; however as mentioned in Section 2.4, we must be careful to assure that the correlation matrix is not ill-conditioned. If for example two networks in the population are very similar, two of the rows of C will be nearly collinear. This collinearity will make inverting the correlation matrix very error prone and will lead to very poor results. Thus in the case of the GEM estimator it is important to remove all duplicate (or nearly duplicate) networks from the population $\mathcal{F}$. Removing duplicates can be easily done by examining the correlation matrix. One can remove all networks for which the dot product of its row in the correlation matrix with any other row in the correlation matrix is above some threshold. This threshold can be chosen to allow a number of nets equal to the number of distinct networks in the population as described in Chapter 4.

An alternative approach (Wolpert, 1990) which avoids the potential singularities in C is to allow a perceptron to learn the appropriate averaging weights. Of course this approach will be prone to local minima and noise due to stochastic gradient descent just as the original population $\mathcal{F}$ was; thus we can train a population of perceptrons to combine the networks from $\mathcal{F}$ and then average over this new population. A further extension is to use a nonlinear network (Jacobs et al., 1991; Reilly et al., 1987; Wolpert, 1990) to learn how to combine the networks with weights that vary over the feature space and then to average an ensemble of such networks. This extension is reasonable since networks will in general perform better in certain regions of the feature space than in others. This approach is especially useful when the weighting can not be solved in closed form as is the case when we consider alternative optimization functions in Chapter 3.

In the case of the BEM estimator, we know that as the population size grows our assumptions on the misfits, $m_i(x)$, are no longer valid. When our assumptions breakdown, adding more nets to the population is a waste of resources since it will not improve the performance and if the nets we add have particularly poor performance, we can actually lower the performance of the BEM estimator. Thus it would be ideal if we could find the optimal subset of the population $\mathcal{F}$ over which to average. We could try all the $2^N - 1$ possible non-empty subsets of $\mathcal{F}$ but for large N this search becomes unmanageable. Instead, we can order the elements of the population according to increasing mean square error [8] and generate a set of N BEM estimates by adding successively the ordered elements of $\mathcal{F}$. We can then choose the best estimate. In this case, the BEM estimator is then guaranteed to be at least as good as the naive estimator.

We can further refine this process by considering the difference between the mean square error for the BEM estimator for a population of N elements and the mean square error for the BEM estimator for the same population plus a new net. From this comparison, we find that we should add the new net to the population if the following inequality is satisfied,

$$(2N+1)\mathrm{MSE}[\hat{f}_N] > 2 \sum_{i \neq \mathrm{new}} E[m_{\mathrm{new}} m_i] + E[m_{\mathrm{new}}^2],$$

[7] This approach is essentially the naive estimator for the population of BEM and GEM estimators. Averaging over the population of BEM or GEM estimators will not improve performance.

[8] The first element in this sequence will be the naive estimator.

where $\text{MSE}[\hat{f}_N]$ is the mean square error for the BEM estimator for the population of N and m_{new} is the misfit for the new function to be added to the population. The information to make this decision is readily available from the correlation matrix, C. Now, if a network does not satisfy this criterion, we can swap it with the next untested network in the ordered sequence.

2.6 Discussion

We have developed a general mathematical framework for improving regression estimates. In particular, we have shown that by averaging in functional space, we can construct neural networks which are guaranteed to have improved performance.

An important strength of the ensemble method is that it does not depend on the algorithm used to generate the set of regressors and therefore can be used with any set of networks. This observation implies that we are not constrained in our choice of networks and can use nets of arbitrary complexity and architecture. Thus the ensemble methods described in this chapter are completely general in that they are applicable to a wide class of problems including neural networks and any other technique which attempts to minimize the mean square error. In Chapter 3, we will show that the averaging framework can be generalized further to include a very broad class of optimization problems.

One striking aspect of network averaging is the manner in which it deals with local minima. Most neural network algorithms achieve sub-optimal performance specifically due to the existence of an overwhelming number of sub-optimal local minima. If we take a set of neural networks which have converged to local minima and apply averaging we can construct an improved estimate. One way to understand this fact is to consider that, in general, networks which have fallen into different local minima will perform poorly in different regions of feature space and thus their error terms will not be strongly correlated. It is this lack of correlation which drives the averaging method. Thus, the averaging method has the remarkable property that it can efficiently utilize the local minima that other techniques try to avoid.

It should also be noted that since the ensemble methods are performing averaging in functional space, they have the desirable property of inherently performing smoothing in functional space (See Section 3.7). This property will help avoid any potential over-fitting during training.

In addition, since the ensemble method relies on multiple functionally independent networks, it is ideally suited for parallel computation during both training and testing.

We are working to generalize this method to take into account confidence measures and various nonlinear combinations of regression estimators. (See Chapter 6)

Chapter 3

Extensions to Convex Optimization

3.1 Introduction

In Chapter 2, we showed that we can generate improved regression estimates for MSE optimization problems using averaging methods. In this chapter we demonstrate the generality of the averaging methods by showing that they can be extended wide class of optimization problems. We show that the convexity property is the key to the power of averaging.

In Section 3.2, we relax the assumption made in Section 2.2 to show that the averaging result holds in general for MSE optimization. We note that MSE optimization is a special case of l_p-norm optimization in Section 3.3 and we show that averaging can be extended to these norms. In Section 3.4, we discuss the notion of convexity and its relation to averaging. In Section 3.5, we use the convexity result to generalize the averaging method to a wide variety of optimization techniques. Section 3.6 discusses non-convex optimization. In Section 3.7 we link the averaging methods to standard regularization techniques by showing that averaging is performing smoothing by variance reduction.

3.2 Removing the Independence Assumption

In general, our assumptions made in Section 2.2, that the misfit functions are independent and have zero mean, will not hold. However, these assumptions can be relaxed. In what follows, we prove that for MSE optimization the averaging methods always produces an improved regression function without the independence and zero mean assumptions.

From the *Cauchy inequality* (Beckenbach and Bellman, 1965),

$$\left(\sum_{i=1}^{n} x_i y_i\right)^2 \leq \left(\sum_{i=1}^{n} x_i^2\right)\left(\sum_{i=1}^{n} y_i^2\right),$$

we have, by setting $y_i = 1 \ \forall \ i$, that

$$\left(\sum_{i=1}^{n} x_i\right)^2 \leq n \sum_{i=1}^{n} x_i^2.$$

If we now replace the x_i with our misfit functions, m_i, and average over the data we find that[1]

$$\text{MSE}[\overline{f}] \leq \overline{\text{MSE}[f]}. \tag{3.1}$$

Equation (3.1) tells us that the average regressor (i.e. the BEM estimate) is always better than the population average. Note however that by relaxing the independence and zero mean assumptions we lose the $\frac{1}{N}$ behavior from Equation 2.2.

This result clearly extends to *generalized least mean squares* (Carroll and Ruppert, 1988b) where for some non-zero function $g(x)$ we minimize

$$\sum_{i=1}^{n}(y_i - f(x_i))^2/g^2(x_i).$$

3.3 Extensions to l_p-Norms

The result of Eqn. (3.1) can be stated more generally: Optimization procedures which seek to minimize an l_2-norm cost function will always benefit from averaging. This is a powerful result due to the fact that most optimization done today is somehow related to a mean square error minimization problem. However since other l_p-norm minimization is not uncommon (Gonin and Money, 1989) particularly for $p = 1$ and $p = \infty$, it is interesting to consider the case when $p \neq 2$.

From *Hölder's inequality* (Gradshteyn and Ryzhik, 1980) for $x_i, y_i \geq 0, 1/p + 1/q = 1$ and $p > 1$,

$$\sum_{i=1}^{n} x_i y_i \leq \left(\sum_{i=1}^{n} x_i^p\right)^{\frac{1}{p}} \left(\sum_{i=1}^{n} y_i^q\right)^{\frac{1}{q}},$$

we find by setting $y_i = 1\ \forall\ i$ and using $|x_1 + \cdots + x_n| \leq |x_1| + \cdots + |x_n|$ that

$$\left|\frac{1}{n}\sum_{i=1}^{n} x_i\right|^p \leq \frac{1}{n}\sum_{i=1}^{n}|x_i|^p \tag{3.2}$$

for all x_i and $p > 1$. Equation (3.2) implies that any l_p-norm minimization procedure with $p \geq 1$[2] will benefit from the application of averaging. In general we have that any cost function of the following form will benefit for averaging:

$$E(\{x_j\}) = \sum_{ij} \alpha_i |x_j|^{p_i},$$

where $\alpha_i \geq 0$ and $p_i \geq 1$.

Note that these results generalize in the natural way to L_p norms.

---

[1] Here and in all that follows, a bar will indicate an average over the population of regression estimates. For example, $\overline{f}$ is defined as

$$\overline{f} \equiv \frac{1}{n}\sum_i f_i(x).$$

[2] The $p = 1$ case does not follow from the argument above; however its proof is trivial.

3.4 Convexity and Averaging

In this section we show that a sufficient condition for averaging to generate improved regression estimates is for the optimization measure to have the convexity property. Convexity is defined in the following way. A function, $h(x)$, is convex on an interval $[a, b]$ if $\forall\ x_1, x_2 \in [a, b]$

$$h\Big(\frac{x_1 + x_2}{2}\Big) \leq \frac{h(x_1) + h(x_2)}{2}.$$

If $\Phi(u)$ is a convex function on the interval $\alpha \leq u \leq \beta$ and $f(x;\omega)$ and $g(\omega)$ defined on $[a, b]$ satisfy $\alpha \leq f(x;\omega) \leq \beta\ \forall\ x$ and $g(\omega) \geq 0$ then *Jensen's inequality* (Gradshteyn and Ryzhik, 1980; Hardy et al., 1952) states

$$\Phi\left(\frac{\int_a^b f(x;\omega)g(\omega)d\omega}{\int_a^b g(\omega)d\omega}\right) \leq \frac{\int_a^b \Phi(f(x;\omega))g(\omega)d\omega}{\int_a^b g(\omega)d\omega}. \tag{3.3}$$

If we use our population of estimators, $\{f(x;\omega_j)\}$, to define

$$g(\omega) \equiv \frac{1}{n}\sum_{j=1}^{n} \delta(\omega - \omega_j),$$

where ω_j corresponds to the parameters of the jth regression estimate, then Eqn. (3.3) becomes

$$\Phi(\overline{f}) \leq \overline{\Phi(f)} \tag{3.4}$$

which is just the discrete version of Jensen's Inequality. Thus, we have the following fundamental result:

Theorem 1 *Given a convex cost function $\Phi : \mathcal{R}^n \mapsto \mathcal{R}$ and a set of functions $f_i : \mathcal{R}^m \mapsto \mathcal{R}^n$ for some n and m, then the cost of the average of the functions, f_i, is always less than or equal to the average of the cost of the individual functions.*

This theorem, as stated, applies only to the BEM estimators; however a natural corollary which extends this result to the GEM estimator is given below.

Corollary 1 *For any convex cost function $\Phi(x)$, the GEM estimator has a lower cost that the BEM estimator, i.e*

$$\Phi(f_{\mathrm{GEM}}) \leq \Phi(\overline{f}) \leq \overline{\Phi(f)}.$$

The proof of this corollary follows directly from the minimization of the cost $\Phi(f_{\mathrm{GEM}})$ relative to the linear weights in the GEM estimator and the observation that the BEM estimator is a special case of the GEM estimator. In the case of the MSE, the cost function is quadratic and we can therefore find the optimal weights in closed form; however for general convex costs we will not be able to solve for the optimal weights in closed form. When closed form solutions do not exist, we can still find approximate solutions by using an iterative root finding algorithm or gradient descent.

With this theorem in hand, we can now go back to the l_p-norms. In the case of l_p-norms, Eqn. (3.4) implies that not only can we use $p \geq 1$ but that we can *not* use $p < 1$ unless we restrict $f(x)$ to be non-negative in the case of $p < 0$ or in the case of $0 < p < 1$ additionally require that we maximize the cost (or minimize the negative cost for $0 < p < 1$. This restriction is not too severe as it still applies to optimizations that deal directly with probabilities. Thus for x_j non-negative, we have a very wide selection of cost functions given by [3]

$$E(\{x_j\}) = \sum_{ij}(\alpha_i x_j^{p_{\alpha i}} - \beta_i x_j^{p_{\beta i}}),$$

where $\alpha_i, \beta_i \geq 0$ and $p_{\alpha i} \in (-\infty, 0) \cap [1, +\infty)$ and $p_{\beta i} \in (0, 1)$. Of course for $p \in (0, 1)$, the negative of the cost function is not bounded below. We can avoid this unboundedness be requiring a cost term with sufficiently large p. Also note that costs with $p < 0$ and negative costs with $0 < p < 1$ weight large errors more lightly than small errors!

3.5 Extending Averaging to Other Cost Functions

In the preceding sections, we have seen that averaging can be applied to a wide variety of cost functions. In this section we show how Maximum Entropy (ME), Maximum Mutual Information (MMI), Maximum Likelihood Estimation (MLE) and the Kullback-Leibler Information (KLI) can all benefit from averaging.

Suppose now that the functions that we are estimating are probability densities given by $p_j(x_i)$ where i is the index of the data and j is the index of the population of density function estimates.[4] In this case, the entropy of $p(x)$ is given by

$$H(p) = -\sum_i p(x_i) \ln p(x_i).$$

Note that $\Phi(z) \equiv z \ln z$ then $\Phi(z)$ is a convex function and therefore we can write for each data point that

$$\frac{1}{n}\sum_j p_j(x_i) \ln p_j(x_i) \geq \Big(\frac{1}{n}\sum_j p_j(x_i)\Big) \ln\Big(\frac{1}{n}\sum_j p_j(x_i)\Big)$$

which implies

$$H(\overline{p}) \geq \overline{H(p)}, \tag{3.5}$$

where the overline indicates an average over the population index j. From Eqn. (3.5), we see that averaging helps to *maximize* the entropy and should therefore be useful in ME optimization (Skilling, 1989; Kapur and Kesavan, 1992). Baldi (Baldi, 1991) suggests that *minimizing* the entropy could also be useful in neural net learning; however averaging should not be used with this method as it would degrade performance. We must keep in mind that averaging will help minimize a convex function while it will help maximize a negative convex function such as entropy.

Mutual Information (Galland and Hinton, 1990; Bridle et al., 1992; Bridle, 1990; Linkser, 1989) can be defined in terms of the entropy as

$$I(a, b) = H(a) + H(b) - H(ab).$$

Therefore since $\Phi(u, v) \equiv u \ln u + v \ln v - uv \ln uv$ is a convex function[5] we can proceed as we did for entropy and to find that

$$I(\overline{p}, b) \geq \overline{I(p, b)},$$

[3] The sum of two convex functions is convex.
[4] Note that since the p_j's are densities, the average over j is also a density.
[5] Consider $\Phi(u, v) = (1 - v)u \ln u + (1 - u)v \ln v$.

where $b(x)$ is some fixed reference density and the overline indicates an average over the population index j. Thus averaging should help MMI optimization.

We now turn to MLE which is probably the most commonly used alternative to LMS optimization. In MLE we attempt to maximize the *likelihood function* (Wilks, 1962),

$$L(p) = \prod_i p(x_i),$$

where $p(x_i)$ is the probability of event x_i. We consider here the equivalent problem of maximizing the log-likelihood function, $\ln L(p)$. Starting with the *arithmetic-geometric mean inequality* (Beckenbach and Bellman, 1965),

$$\frac{a_1 + \cdots + a_n}{n} \geq (a_1 \cdots a_n)^{1/n},$$

where $a_i \geq 0 \; \forall \; i$, we substitute $p_j(x_i)$ for a_i, take the log of both sides and sum over all of the data to get

$$\sum_i \ln(\frac{1}{n} \sum_j p_j(x_i)) \geq \frac{1}{n} \sum_{ij} \ln(p_j(x_i)),$$

where i is the data index and j is the population index. This result can be re-written as

$$\ln L(\overline{p}) \geq \overline{\ln L(p)}, \tag{3.6}$$

where the overline indicates an average over the population index j. Equation (3.6) demonstrates that averaging always increases MLE.

Finally, KLI is a commonly used measure in density estimation (Kullback and Leibler, 1951; Härdle, 1990; Devroye, 1987) and has been shown to be useful in neural network hand-written character recognition (Xu et al., 1990). The KLI is given by

$$K(f, g) = \int f \ln(\frac{f}{g}),$$

and can be thought of as a distance between the two probability densities, f and g. Note that the KLI is not symmetric in f and g and is therefore not a proper distance. It can easily be symmetrized but this is rarely done. The KLI is sometimes know as the Cross Entropy or the Relative Entropy (Bridle, 1990). It has been shown that minimizing KLI subject to appropriate constraints leads to optimal learning rules (Qian et al., 1991).

From Equation (3.3) we have that $-\int f \ln(\frac{f}{g}) \leq \ln(\int f \frac{g}{f}) = 0$. Thus $K(f, g) \geq 0$. So we have that $K(p_i, \overline{p}) \geq 0$ which can be re-written as

$$\int p_j \ln(\frac{p_j}{g}) \geq \int p_j \ln \frac{\overline{p}}{g}$$

for some probability density g. If we sum over the population index j, we find

$$\overline{K(p, g)} \geq K(\overline{p}, g).$$

So the Kullback-Leibler Information between probability densities is reduced by averaging.

3.6 Nonconvex Cost Functions

In general, averaging will not be helpful when the cost function to be minimized is not convex. Non-convexity is the case, for example, when we consider cost functions which saturate at high values such as cost functions which attempt to minimize the effect of outliers (see Fig. 3.1). However for cost functions with one convex region, one can determine the percentage of data points that lie in the convex region. If this percentage is sufficiently high, averaging may still be beneficial.

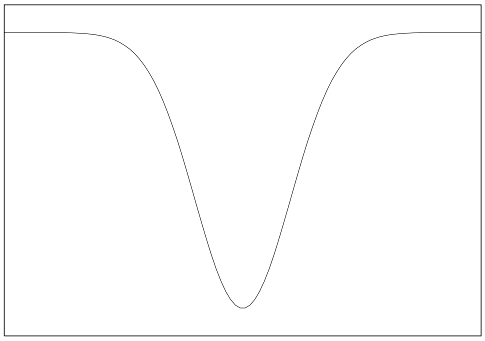


Figure 3.1: Example of a cost function which saturates at high values.

3.7 Smoothing by Variance Reduction

The averaging process is inherently a smoothing operation as can be seen by the following derivation. Fixing x, we can write that

$$\begin{aligned} E[(\hat{f}(x) - f(x))^2] &= E[(\hat{f} - E[\hat{f}] + E[\hat{f}] - f(x))^2] \\ &= E[(\hat{f} - E[\hat{f}])^2] + E[(E[\hat{f}] - f(x))^2] \\ &\quad +2E[(\hat{f} - E[\hat{f}])(E[\hat{f}] - f(x))] \\ &= \mathrm{VAR}(\hat{f}) + \mathrm{BIAS}^2(\hat{f}) \end{aligned}$$

where $\mathrm{VAR}(\hat{f}) \equiv E[(\hat{f} - E[\hat{f}])^2]$ and $\mathrm{BIAS}(\hat{f}) \equiv E[\hat{f}] - f(x)$. Now since $E[\overline{f}] = E[\hat{f}]$, we have that the bias term is the same for both $\mathrm{MSE}[\hat{f}]$ and $\mathrm{MSE}[\overline{f}]$ and therefore when we reduce the MSE by averaging it is because we have reduced the variance term. This variance reduction corresponds to smoothing our estimate of $\hat{f}$.

Further, we observe that in the infinite limit the sum $\overline{f} = \frac{1}{n}\sum_i \hat{f}_i$ converges to $E[f]$, i.e.

$$\lim_{n\to\infty} \overline{f}(x) = \int f(x;\beta)p(\beta)d\beta$$

where β represents the weights of a network and $p(\beta)$ is the probability that our neural net generating process will generate a net at point β. Thus we can make the variance term as small as we like and we are essentially left only with the bias term. Note that if $p(\beta)$ corresponded to the true distribution of β and if $f(x;\beta)$ corresponded to the true model, then the averaging is equivalent to Monte Carlo Integration (Kalos and Whitlock, 1986) and Bayesian Inference (Duda and Hart, 1973).

3.8 Penalized MLE, Smoothing Splines and Regularization

Averaging can also be used with methods which explicitly try to avoid over-fitting through the use of penalty terms (Barron, 1991; Poggio and Girosi, 1990; Rissanen, 1986). In the case of penalized MLE and smoothing splines (Härdle, 1991; Hastie and Tibshirani, 1990; Wahba and Wold, 1975) the penalty term takes the form of a regularizer which attempts to measure a solution's smoothness,

$$\ln \prod_i p(x_i) + \lambda \int (p'')^2 dx$$

$$\frac{1}{n} \sum_i (f(x_i) - f_i)^2 + \lambda \int (f'')^2 dx.$$

The smoothing parameter, λ, regulates how much smoothing is performed and can be estimated using cross-validation. Other regularizers can be used and in general they take the form

$$\int (D(p))^2 dx$$

where D is some linear differential operator corresponding to some *a priori* knowledge of the system to be fit. Regularizers of this form are convex; therefore averaging will reduce the regularizer penalty and therefore increase the smoothness of the solution.

Chapter 4

Experimental Results

In this chapter, we present experimental results on three real-world classification and time series prediction tasks comparing the performance of MLPs with and without the use of the Basic Ensemble Method. The three databases used were the NIST OCR database; the Turk and Pentland Human Face database; and the time series of sunspots from the year 1700 to 1979.

4.1 Neural Network Regressors

Consider the standard regression problem

$$y = f(x) + n$$

where y is a random variable with mean $f(x) = E[y|x]$ and n is independent zero-mean noise.

Classification was performed using non-parametric regression estimates in the form of multilayer perceptrons (MLP). The MLP is a special case of additive model performing projection pursuit (Hastie and Tibshirani, 1990). The functional form is given by (Rumelhart et al., 1986)

$$f_k(x; \alpha, \beta) = \sigma(\alpha_{0k} + \sum_{j=1}^{N} \alpha_{jk}\sigma(\beta_{0j} + \sum_{i=1}^{d} \beta_{ij} x_i))$$

were σ is a fixed ridge function chosen to be $\sigma(x) = (1 + e^{-1})^{-1}$; N is a measure of the complexity of the class of functions to which $f_k(x)$ belongs; k is the index of the class for which $f_k(x)$ is an indicator; d is the dimensionality of the data space; and α and β are adjustable parameters.

In the experiments presented here, the following procedure was followed: For each data set, several populations of 10 MLP regression estimates were generated using stochastic gradient descent (Werbos, 1974) to minimize the empirical mean square error. Each population had a fixed N which was allowed to vary from population to population. Each regressor was initialized with a different random set of parameters. The gradient descent was stopped using a cross-validatory stopping criterion to avoid over-fitting. All results are on independent testing sets.

4.2 Classification Data

The NIST database is a standardized optical character recognition (OCR) database developed by the National Institute for Standards and Technology (NIST). The database was generated by approximately 2000 federal government employees using a standardized form for recording both isolated and contiguous letters and numerals. The employees were chosen to represent as closely as possible the true population distribution of United States writing styles. The forms were scanned into black and white pixel data and segmented according to existing boxes on the forms. For the experiment reported in this chapter, isolated letters and numerals were hand-labelled; normalized to a fixed size; and convolved with a Gaussian filter to smooth character edges. The dimensionality of the images was reduced by convolving each filtered image with edge-detecting kernels of various orientations.[1] The NIST database was divided into three groups (numerals, uppercase letters and lowercase letters). This division was performed due to time and computer resource constraints.

A sample of the preprocessed NIST data is shown in Figure 4.1. Note that after preprocessing, the characters are no longer discernible. The MLPs were trained using these images as input.

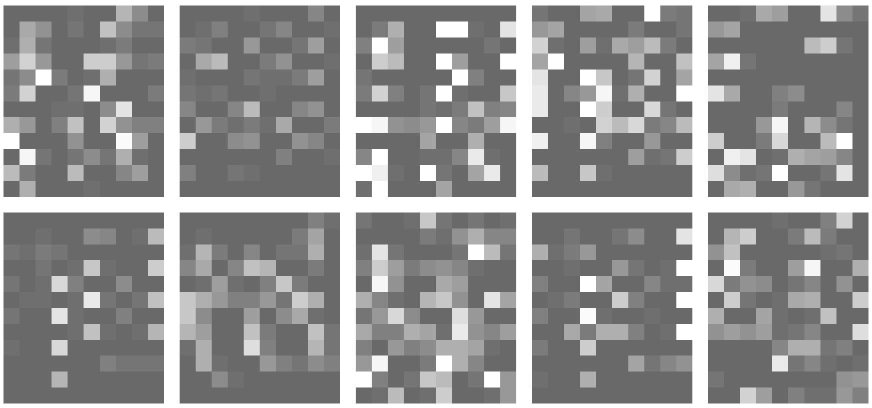


Figure 4.1: Preprocessed NIST data representing the digits '0' through '9'.

A subset of the preprocessed human face data is shown in Figure 4.2. In this figure, one can see artifacts of the preprocessing. In some of the corners, there are black triangles from rotating and some of the faces may appear slightly stretched from "warping". These images were used as inputs to the MLPs during training.

The human face database was created by Turk and Pentland at the MIT Media Lab. It was further processed by Reisfeld et al. (1992) so that the location of eyes and tips of the mouth were fixed using a symmetry operator (Reisfeld and Yeshurun, 1992) and background eliminated.[2] It consists of multiple images of 16 different male faces. The images were generated under various lighting conditions and with various locations and orientations of the faces.

Each database was divided into three independent sets (training, testing and cross-validatory). The data set statistics are summarized in Table 4.1.

For each data set, several populations of 10 MLP networks were trained. The MLP networks all had a single hidden unit layer and each population had a fixed network architecture; however, the number of hidden units in each network was allowed to vary between different populations. Each network was initialized with a different random configuration of weights. Training was stopped using a cross-validatory stopping criterion. All results reported are on the independent testing sets unless otherwise specified.

[1] The majority of this preprocessing work was performed at Nestor, Inc., Providence, RI.

[2] Thanks to Daniel Reisfeld for making this preprocessed database available to the Brown University Institute for Brain and Neural Systems.

DATA SET	DIM	TRAINING SET	CV SET	TESTING SET	CLASSES
Numbers	120	13241	13241	4767	10
Uppercase	120	11912	11912	7078	26
Lowercase	120	12971	12970	6835	26
Faces	2294	136	135	160	16

Table 4.1: The table shows the dimensionality, the number of classes and the breakdown of the data into various independent sets.

4.3 Performance Criteria

There are three criteria by which we will measure network performance. They are the MSE, the percent correct and the Figure of Merit (FOM).

The MSE is the most fundamental measure of the three and is specifically what we set out to minimize; so we present results demonstrating that the MSE is reduced through the use of averaging.

Minimizing the MSE is sufficient if we are only concerned with regression; however in classification problems, one is usually only interested in minimizing the MSE so far as it minimizes the percent incorrect classification. It is important to note that, although most classification algorithms attempt to optimize classification performance by minimizing the MSE, we have no guarantee that there is a one-to-one correspondence between the MSE and the percent incorrect. In fact, we know that the MSE is a continuous measure while the percent correct is discrete. Also it is possible to construct an estimator with a very high MSE but perfect classification performance.[3] or even worse, an estimator which will reduce the MSE while increasing the number of classification errors relative to some other estimator. With all of this said, it may seem that there is no hope for improving the classification performance through averaging. In practice however, there is a close correspondence between minimizing the MSE and maximizing the classification performance. We demonstrate this fact by presenting results showing that percent correct classification performance also is improved through averaging.

If we want to be even more practical, we should consider the goal of our classification task. Typically for many classification tasks it is worse for the network to make an error than it is for the network to reject a pattern. For example, if a machine is reading postal zip codes and it incorrectly classifies a '9' as a '0', a package could be sent to the wrong coast. This error costs much more than if the machine could reject patterns that it is uncertain about and send those special cases to a human. A much more dramatic example can be found in medical diagnosis where a false negative is much worse than a false positive! Thus, it is common that a network that rejects nothing but makes mistakes is less desirable than a network which rejects some patterns but makes fewer mistakes. Perhaps the simplest method for taking the various costs of classification into account is to weight the rejects and the errors by their relative costs. We call such a weighted criterion a Figure of Merit (FOM).

The United States Postal Service defined the following FOM for zip code numeral recognition:

$$\text{FOM} \equiv \%\text{Correct} - \%\text{Rejected} - 10(\%\text{Error}).$$

[3] Consider an MLP for which the sign of the output units is always correct but the output values are close to zero not 1 and -1.

We use this FOM to measure the performance on both the numeral and the letter databases. We should again note that the averaging methods only claim to minimize the MSE and say little about minimizing a FOM. However as with classification performance, we find that the FOM is sufficiently closely linked to the MSE that averaging results in improved FOM performance.

One technique that could be employed to make the connection between the FOM and the MSE closer is to simultaneously minimize three MSEs - one for patterns which are rejected, one for patterns which are incorrectly classified, and one for all patterns - and weight them according to their relative costs.

4.4 Confidence Measure

In order to calculate a FOM for an estimator, a criterion must be chosen for rejecting inputs. This section describes the rejection criterion.

Considering each network output as a distinct model for the generation of the data, the output value can be interpreted as the probability that a particular model generated a particular input (See Appendix A). This interpretation allows us to use statistical inference to calculate a likelihood value. The likelihood that model i generated the input and all the other models did not is given by, $C_i(x)$,

$$C_i(x) = p_i(x) \prod_{j \neq i} (1 - p_j(x)),$$

where x is the input pattern and $p_k(x)$ is the probability that model k generated x. We interpret $C_i(x)$ as our confidence that the input belongs to class i.[4] The highest confidence for a pattern will be for the class with the largest network output.

It was empirically found sufficient to use only the two largest network outputs in the product over i. In order to treat the MLP outputs as probabilities, we scale them between 0 and 1.

A confidence measure was calculated for each pattern in the cross-validation set and a "Confidence Threshold" was chosen to minimize the FOM over the cross-validation set. Patterns with confidence measures equal to or above the confidence threshold were classified. Patterns with confidence measures below the confidence threshold were rejected.

This rejection criterion is easily extended to averaged networks.

4.5 Human Face Recognition

In this section we present results comparing the performance of averaged and unaveraged MLPs trained on the human face database. In Figure 4.3, the MSE is plotted as a function of network size. In this figure and in all of the figures comparing averaged and unaveraged MLP performance the average performance and variance of the networks in a population; the performance of the best individual network from a populations; and the performance of the averaged network are plotted. The average performance of a population is denoted by the line labelled "Individual" since this is the expected performance of any individual net chosen at random. This line also has error bars. The performance of the best individual network from a population is denoted by the line labelled "Best Individual". The performance of the averaged network is denoted by the line labelled "Ensemble". Note that each data point corresponds to a population of networks all with the same number of hidden units.

From Figure 4.3, we see that averaging can lead to dramatic improvements in network performance. Also, it is very interesting to notice averaging over the simplest architectures

[4] N. Intrator, Private communication.

leads to a very significant improvement in performance. We will see this effect over and over again as further results are presented. In some cases the ensemble performance of the least complex MLPs will be comparable to the performance on the most complex architectures. This fact suggests a new learning algorithm: Instead of training complex networks, train a population of simple networks. This approach has two major advantages. First, it accelerates learning since n MLPs with m hidden units train faster than 1 MLP with nm hidden units and because we must train a population of complex MLPs in order to select the best one. Second, less complex nets are less prone to over-fitting the noise in the data than are complex nets.

In Figure 4.4, the percent correct performance on the face data is plotted versus network size. As noted in Section 4.3, we are not specifically optimizing the percent correct; however, it is clear from Figure 4.4 that the averaged network is outperforming the individual networks. Note also that the averaged network is not only outperforming the average performance of the population but is outperforming the best individual network from each population.

It should be emphasized that these results are for an independent test set on which the networks were not trained. Thus, the averaging method is helping to improve the networks' ability to generalize to previously unseen data.

In a similar experiment (Golomb et al., 1991), 90, rotated, scaled and cropped human face images (45 of each gender; 10 different faces) were used to train a network to identify only the gender of the person in the image. The testing error reported for gender classification was 8.1 percent. Our testing error of 1.8 percent for 16 class distinctions compares favorably. In another experiment (Cottrell and Metcalfe, 1991), an error rate of 1 percent was quoted for recognition of 20 different classes of faces; however this task had the advantages that no rotating, scaling, cropping or warping were necessary since all of the subjects were photographed in a fixed location with fixed lighting. In addition, the images in this last experiment were normalized to have equal brightness and variance.

As noted in Section 2.6, although the results presented in this section and in Section 4.6 are based on populations of MLPs with identical architectures, there is nothing to constrain us to average only over networks with the same architecture or networks with the same learning rule or even networks trained on the same data! This line of reasoning leads us to a very rich area of research the simplest step of which is to combine networks of varying architectures. This direction has been left for future research.

4.6 Optical Character Recognition

MSE performance results are shown in Fig. 4.5, Fig. 4.8 and Fig. 4.11.

In these simulations we found an optimal rejection threshold for each network based on the cross-validatory set. FOM results are shown in Fig. 4.7, Fig. 4.10 and Fig. 4.13. Again notice that in all of these results, just as in the straight classification results, the ensemble estimator was not only better than the population average but it was also better than the naive estimator.

Percent correct classification results are shown in Figures 4.6, 4.9 and 4.12. In these plots, the classification performance of the BEM estimator (labelled "Ensemble"), the naive estimator (labelled "Best Individual") and the average estimator from the population (labelled "Individual") are plotted versus the number of hidden units in each individual network. Error bars are included for the average estimator from the population. In all of these plots there is an increase in performance as the number of hidden units increases. Notice however, that in all of these results the ensemble estimator was not only better than the population average but it was also as good as or better than the naive estimator in every case.

These results for difficult, real-world classification tasks show that the BEM estimator is

EXPERIMENT	DATA SET	TEST FOM
Perrone (1992)	NIST Numerals	94.7
Hansen et al. (1992)	NIST Numerals	83.0
Drucker et al. (1992)	NIST Numerals	92.0
Scofield et al. (1991)	NIST Numerals	89.7
Xu et al. (1992)	USPS Numerals	95.0
Denker et al. (1989)	USPS Numerals	76.0
Le Cun et al. (1990,1989)	USPS Numerals	81.0
Fontaine & Shastri (1992)	USPS Numerals	75.4
Drucker et al. (1992)	USPS Numerals	57.4
Martin & Pittman (1990)	Martin Numerals	80.0
Guyon et al. (1989)	Guyon Numerals	85.0[6]
Perrone (1992)	NIST Uppercase	68.1
Drucker et al. (1992)	NIST Uppercase	72.9
Martin & Pittman (1990)	Martin Letters	70.0
Guyon et al. (1990)	Guyon Letters	92.0
Perrone (1992)	NIST Lowercase	70.8
Drucker et al. (1992)	NIST Lowercase	-2.0[7]

Table 4.2: Comparison of OCR results

significantly and dramatically better than standard techniques. Table 4.2 compares the performance of the Ensemble Estimator with various other OCR results.[5] In the table, four different databases are listed: The NIST data has already been described. The USPS data is a standard database scanned directly from zipcodes on dead letters in the U.S. Postal Service. The Martin database was both scanned from real bank checks and entered by 110 subjects using a computer stylus digitizer. The Guyon Database was generated from 10 subjects all trying to mimic one particular writing style which would tend to make this database more uniform than the others. This fact may explain the surprisingly good performance on the Guyon Letters. Note that our results are comparable to the best in every category for the NIST data. It is surprising that Drucker's results have such a wide variability. We have no explanation for this.

Other researchers (Keeler et al., 1991; Matan et al., 1992; Keeler and Rumelhart, 1992; Martin and Rashid, 1992) have attempted to perform classification and segmentation of characters simultaneously. Our results are better than any of the segmentation results however the segmentation/classification task is much more difficult.

In addition to the results presented in this section, the effect of averaging on the rejection

[5]Unfortunately, the results are not all for the same data.

[7]This value had to be read from a graph.

[7]This value is extremely low. It corresponds to an error rate of 8.1 percent and a rejection rate of 29 percent. Drucker et al. (1992) quote this results as being better than their single network which have a FOM of -19 (9.8 percent error and 21 percent rejection).

NUMBER OF HIDDEN UNITS	INDIVIDUAL	BEM
10	92.1 ± 0.6	93.8
16	93.2 ± 0.6	94.6
22	93.8 ± 0.5	94.5
10, 16, 22	93.0 ± 0.9	94.1
16, 22	93.5 ± 0.6	94.6

Table 4.3: Averaging over various architectures: Test data FOM for NIST Numeral Data

and error rates was examined for the FOM calculates.[8] For each NIST data set, the rejection rate was uniformly reduced by averaging and in each case outperformed the population average. For the error rates, the improvement after averaging was not as clear. The ensemble error rate was frequently within one standard deviation of the population average and in two cases was equal to or worse than the population average. It is not clear how to interpret these results. It is possible that the error rate is very close to the optimal error rate in which case we would expect fluctuations around the expected minimum. It is also possible that we are seeing the effect of the fact that we are minimizing the MSE and measuring the FOM. The second possibility seems more likely since no special effort was made to optimization the MLPs beyond the cross-validation stopping rule.

As stated in Chapter 2, one of the advantages of the averaging methods is that there is no constraint on the functional form of the regression estimators; therefore it is also possible to combine networks with various architectures. We test this idea using all of the networks from the populations of 10, 16 and 22 hidden unit networks trained on the NIST numeral data. These results are compared in Table 4.3. There are two reasons for the disappointing performance of the average over different architectures. First, we have shown the FOM performance which is not what the BEM estimator is designed to reduce. If we compare the mean square errors of the estimators we find that the average over different architectures is better than the average of the other mean square errors.[9] Second, the performance of the nets with 10 hidden units is considerably lower than that of the other two architectures and therefore pulls the overall performance down. This fact can be seen from the last row of the table where the networks with 10 hidden units have been removed.

In Table 4.4, we compare the performance on the NIST test data of the BEM and GEM estimators. Initial attempts to calculate the GEM estimator directly from the inverse of the empirical covariance matrix, $\hat{C}_{ij}$, ran into difficulties due to instabilities in the inversion process. Therefore, the results presented in Table 4.4 were calculated under the assumption that the errors made by different regression estimates are uncorrelated. Each GEM weight was the inverse of its corresponding diagonal term in the correlation matrix. The table shows that there is a slight improvement when using GEM.

[8] In order to save space, graphs for the rejection and error rates have been omitted in favor of a brief summary of the results.

[9] The average of the MSE of the three BEM estimators was greater than the BEM over all architectures:

$$\frac{1}{3}(0.010394 + 0.009267 + 0.009169) > 0.009363.$$

These numbers are the MSEs divided by the number of data points and the number of output units.

DATA SET	HIDDEN UNITS	BEM FOM	GEM FOM
Numerals	4	90.6	90.8
Numerals	10	93.8	94.0
Numerals	16	94.6	94.7
Numerals	22	94.5	94.5
Uppercase	10	63.8	63.8
Uppercase	20	67.5	68.0
Uppercase	30	67.1	67.1
Uppercase	40	68.1	68.0
Lowercase	10	60.3	60.7
Lowercase	20	68.4	68.1
Lowercase	30	68.9	69.2
Lowercase	40	70.7	70.8

Table 4.4: Comparison of BEM and GEM estimators' test FOM for the NIST Data

4.7 Counting Local Minima

If averaging 10 estimators is good, perhaps averaging 20 estimators is better. In this section, we examine this idea by considering how the performance of the ensemble estimator depends on the number of networks used in the ensemble.

If we take the BEM result seriously (Eqn. 2.2), we should expect that increasing the number of networks in the population can only improve the BEM estimator. However as stated in Sec. 2.2, eventually our assumptions on the misfits breakdown and Eqn. 2.2 is no longer valid. This fact is clearly demonstrated in Fig. 4.14 where we show the FOM performance saturates as the number of nets in the population increases. In the figure, we see that saturation in this example occurs after only 6 or 8 nets are in the ensemble population.

This is a very interesting result because it gives us a measure of the number of distinct nets in our population which allows us to efficiently use the resources available to us. By "distinct", we mean that the misfits of two nets are uncorrelated.[10] For a more concrete definition, we can define a population of distinct networks as one for which the related correlation matrix, C from Eqn. 2.5, has a robust inverse. In practice we can also define a population of distinct networks as a population for which the ensemble performance drops by a statistically significant amount when one or more members of the population are removed. This definition has the advantage that it avoids inverting C. We would like to avoid inverting C because the inverse may be non-robust for two reasons which we need to distinguish between and can not. The inverse will be non-robust whenever there is a near collinearity between two or more of the rows of C. This collinearity may be due to networks which are very similar or it may be due to noise in our estimate of C. In the first case we have non-distinct nets; in the second case we may have distinct nets but can not tell.

This result also suggests a very important observation: Although the number of local minima in parameter space is extremely large, the number of distinct local minima in functional space

[10] Their output activities are uncorrelated as we vary from one region of the input space to another. We can also define "nearly distinct" to mean output activities which are weakly correlated.

is actually quite small! This result is contrary to the prevailing neural network dogma which states that the number of distinct local minima even for some of the simplest problems is extremely large. Certainly, the number of local minima in parameter space is on the order of $n!$ where n is the number of distinct hidden units. This fact can be seen by considering that any permutation of the hidden units will not affect the functional form of the network. If there are n hidden units then there are $n!$ such permutations. Thus each local minimum has an $n!$-fold degeneracy in weight space; however, each of these points is mapped onto the same point in function space therefore it serves no purpose to think of these points as distinct. Beyond this factorial degeneracy, there still may exist many distinct points in function space but from Fig. 4.14 it is clear that these distinctions are not significant. One way to interpret this result is to think of each local minimum in parameter space mapping to a point in function space and the multitude of points in parameter space are mapping to only a few tightly packed clusters of points. In practice, we are concerned with the functional fit and therefore only need to worry about the number of local minima in function space.

We can make another important observation if we compare Fig. 4.14 with Fig. 4.7. Consider the value of the FOM on the test data for an ensemble of 4 networks (Fig. 4.14). Compare this value to population average FOM for nets with 40 hidden units (Fig. 4.7). These values are not significantly different; however, training a population of large nets ($>$ 40 hidden units) to find the best estimator is computationally much more expensive than training and averaging a population of small nets. In addition, small networks are more desirable since they are less prone to over-fitting than large networks.[11]

It is also interesting to note that there is a striking improvement for an ensemble size of only two. This observation is even more interesting in light of the fact that the order in which the networks used in Fig. 4.14 were averaged was chosen at random.

4.8 Regression Data

Yearly sunspot statistics have been gathered since the year 1700. We used a subset of this data corresponding to the yearly average sunspot activity from 1700 to 1979.[12] The data is plotted in Figure 4.15. The numbers plotted correspond to a statistic defined by $k(10g + f)$ where g is the number of sunspot groups, f is the number of individual sunspots, and k is a scale factor to normalize different telescopes' (Marple, 1987).

4.9 Time Series Prediction

Since the sunspot data has been extensively studied and has served as a benchmark in the statistics literature (Weigend et al., 1990; Priestley, 1981), we have chosen to study it in the context of averaged neural network regression estimates.

Eleven populations of 10 networks were trained. Each network of a population had the same number of hidden units and the number of hidden units of populations varied from 0 to 10. Each network had 12 inputs corresponding to 12 consecutive yearly sunspot activities and 1 output which was trained to correspond to the subsequent year's sunspot activity. The breakdown of the data is given in Table 4.5. Unlike the classification experiments in Sections 4.5 and 4.6, the time series data has a definite order. Therefore patterns for the training, CV and

[11] Of course we can not make the individual nets too small (in terms of the number of hidden units) or they will not have sufficient complexity and recognition performance will suffer. It was found experimentally that nets with less than approximately 10 hidden units could not adequately perform classification on the NIST Uppercase data.

[12] Special thanks to Andreas Weigend for making this data available to us.

DATA SET	DIM	TRAINING SET	CV SET	TESTING SET
Sunspots	12	90	89	89

Table 4.5: Sunspot Datasets

testing were not chosen randomly. Instead, these sets were constructed such that each set was a continuous segment of time. The training set was the beginning third of the time series, the CV set was the middle third and the testing was the last third.

The MSE performance of the trained networks is shown in Figure 4.16. Note first that the BEM estimate consistently performs better than the population mean although not dramatically so. The next thing to notice is that the GEM performance is extremely variable. This problem is due to two facts: The data set has too few points to generate a reliable estimate of the correlation matrix; and similarity in the functional form of many of the networks caused collinearity in the correlation matrices leading to non-robust inverses. This problem is particularly acute in the population with 3 hidden units which suggests that this population had the fewest local minima.[13] A simple subsampling process was implemented to remedy this problem. The rows of the correlation matrix were compared. If two rows were too similar, one of the corresponding networks was randomly chosen and removed from the population. The "subsampled" points for the 3 hidden unit population are shown for the BEM and GEM estimate using only 5 networks. It is clear from the dramatic improvement in the GEM value that collinearity in the correlation matrix is a major source of variability of the GEM estimate. Even with these problems, the GEM estimate provides the lowest MSE error over all.

We now compare the results presented here with previous work on this data. The results are presented in terms of the average relative variance (ARV) which is defined as the MSE divided by the variance of the data set.

A recent survey of sunspot prediction models (Priestley, 1988) favors the Threshold Autoregressive model (TAR) of Tong and Lim (1980,1983). The TAR model is a combination of two linear autoregressive models with an activity threshold above which one autoregression model is used and below which the other is used (Tong, 1990). One possible TAR function is given by

$$x_i = \begin{cases} \alpha_0 + \sum_{j=1}^{n} \alpha_j x_{i-j}, & x_{i-1} \geq \theta \\ \beta_0 + \sum_{j=1}^{h} \beta_j x_{i-j}, & x_{i-1} < \theta \end{cases}$$

where the α's, β's and θ can be adjusted. Tong and Lim found that setting $n = 12$ gave the best results for the sunspot data. This value for n was adopted for all our simulations.

More recent work performed by Weigend (Weigend et al., 1990) used the neural net model proposed by Lapedes and Farber (1987). The network model had the standard multilayer perceptron architecture with 12 input units 8 hidden units and 1 unsigmoided output unit and is given by

$$f(x) = \beta_0 + \sum_{j=1}^{8} \beta_j \sigma(\alpha_{0j} + \sum_{i=1}^{12} \alpha_{ij} x_i)$$

where $\sigma(x) = (1+e^{-x})^{-1}$ and the α's and β's are adjustable. The method of weight elimination

[13]It is interesting to note that this population has the architecture which Weigend found to be optimal (Weigend et al., 1990).

METHOD	TRAINING ARV	TESTING ARV
TAR Estimator	0.097	0.097
Weight Elimination	0.082	0.086
GEM Estimator w/o CV	0.080	0.084
GEM Estimator w/ CV	0.124	0.124

Table 4.6: Average relative variances for Sunspot Data

(Rumelhart, 1988) was used to avoid overfitting. This method minimizes

$$\mathrm{MSE}[f_\lambda(x)] + \lambda \sum_i \frac{w_i^2}{w_0^2 + w_i^2}$$

where the w_i's represent all of the adjustable parameters of the model, w_0 is the scale factor, and λ is the smoothing parameter.

The results from these models and the GEM estimator are presented in Table 4.6. The training set for the first three models was the data from 1700 to 1920 and the testing data for all four models was 1921 through 1959. The last model in the table corresponds to the best estimator from Figure 4.15 (GEM for 6 hidden unit population) which was trained on only the first 90 years of sunspot activity data. The third model in the table was a GEM estimate from a population of 10 networks with 3 hidden units trained with no cross-validation set.

The first thing to note from the table is that the GEM estimate without cross-validation has a lower ARV than any of the other models. The next thing to notice is that the GEM estimate with cross-validation is worse than any of the other models! This is not surprising since this estimator was trained with less than half as much data as the other three models. In addition, the process generating the sunspots is known to be non-stationary (Weigend et al., 1990); thus testing becomes increasingly unreliable as it moves further into the future away from the training data.[14] For the first three models in the table, the testing and training data is contiguous in time; while for the GEM model with cross-validation, there is more than 100 years of sunspot activity data between the training and testing data. In light of these facts, it is impressive that the GEM estimator with cross-validation is as good as it is.

[14] The non-stationarity of the sunspot process motivated our choice of presenting results from nets trained with cross-validatory stopping in Figure 4.16. Separating the training and testing data in time makes the prediction problem more difficult.

Figure 4.2: Preprocessed human face data representing 16 different male faces.

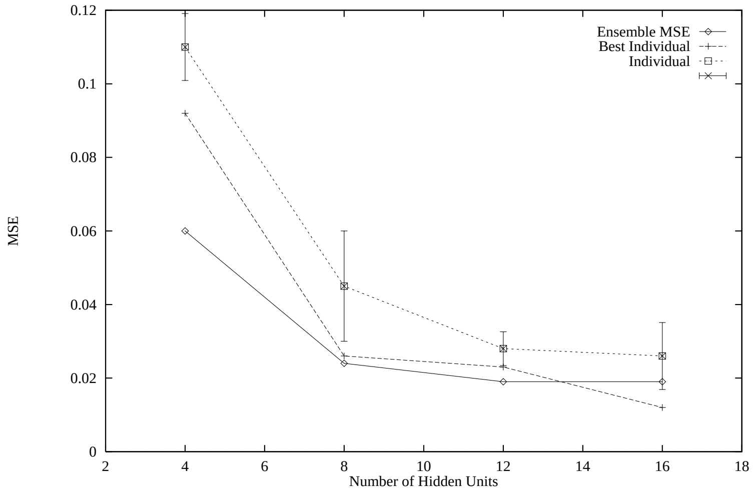


Figure 4.3: Human Face Data: MSE vs. Network Size

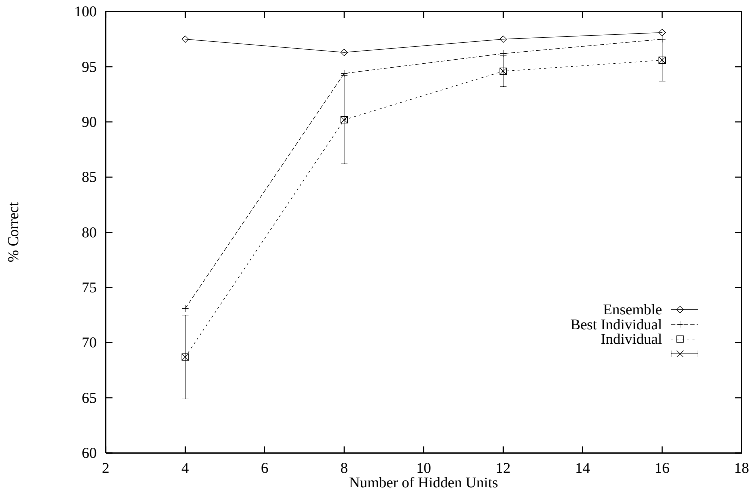


Figure 4.4: Human Face Data: Percent Correct vs. Network Size

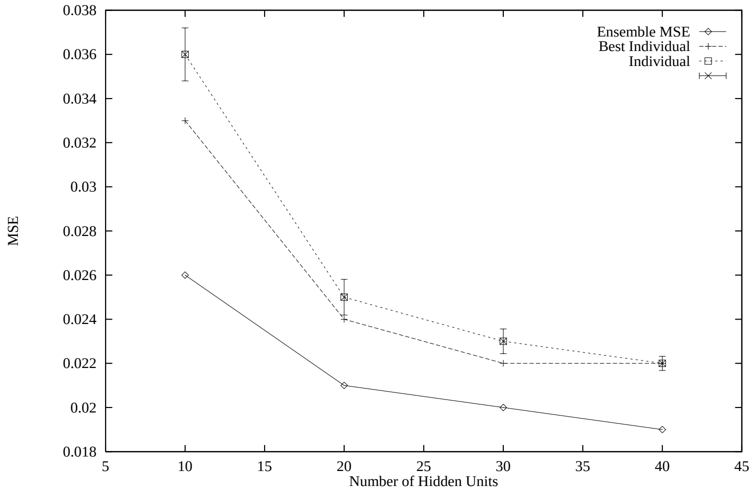


Figure 4.5: NIST Uppercase Data: MSE vs. Network Size

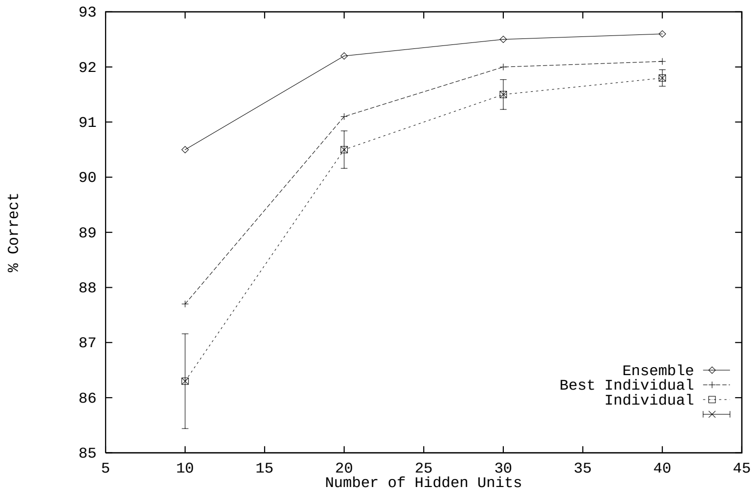


Figure 4.6: NIST Uppercase Data: Percent Correct vs. Network Size

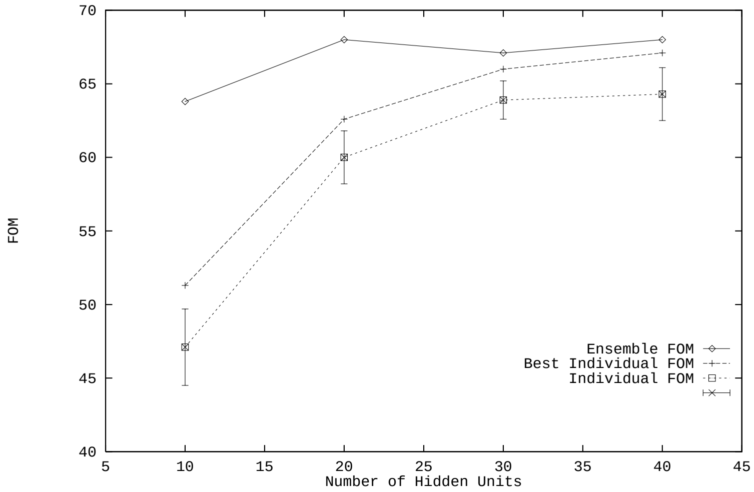


Figure 4.7: NIST Uppercase Data: FOM vs. Network Size

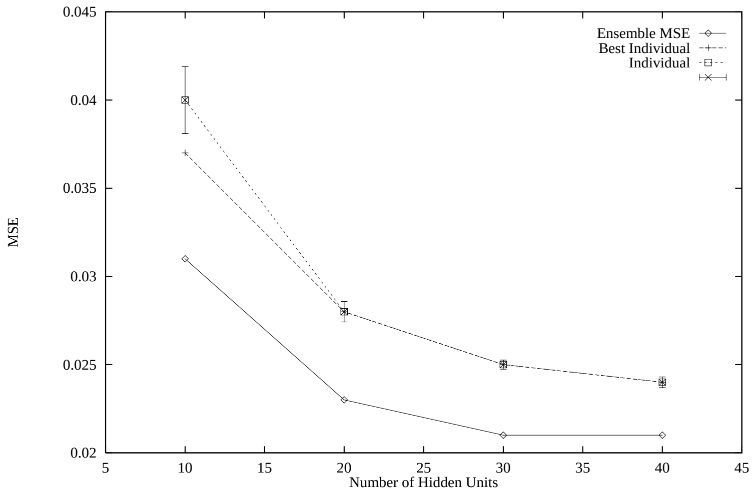


Figure 4.8: NIST Lowercase Data: MSE vs. Network Size

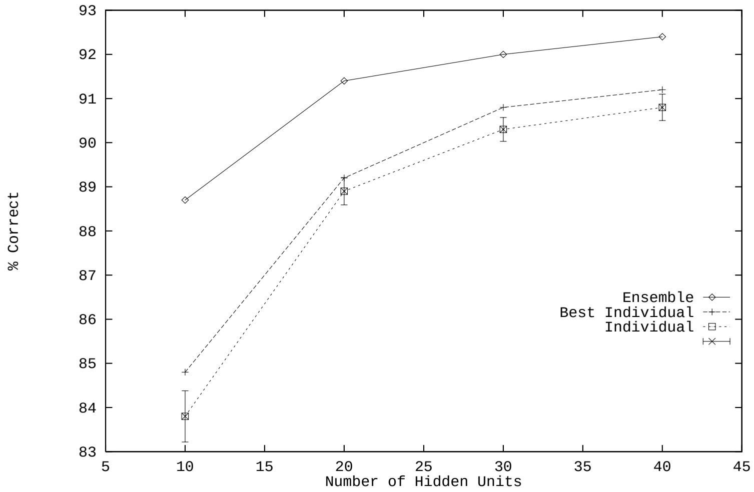


Figure 4.9: NIST Lowercase Data: Percent Correct vs. Network Size

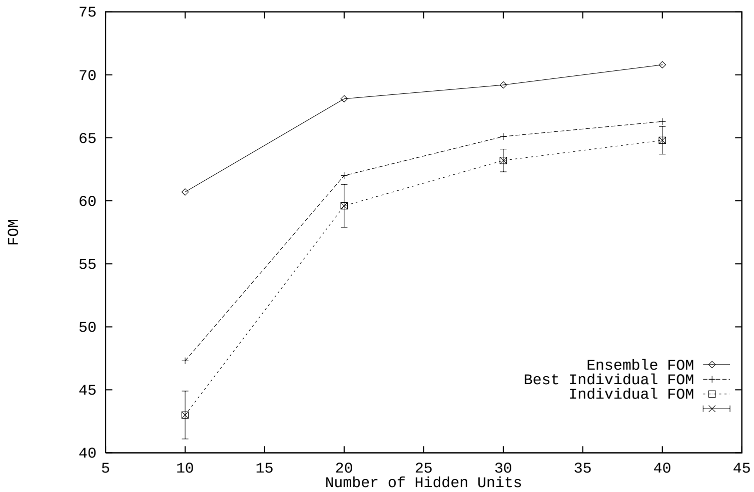


Figure 4.10: NIST Lowercase Data: FOM vs. Network Size

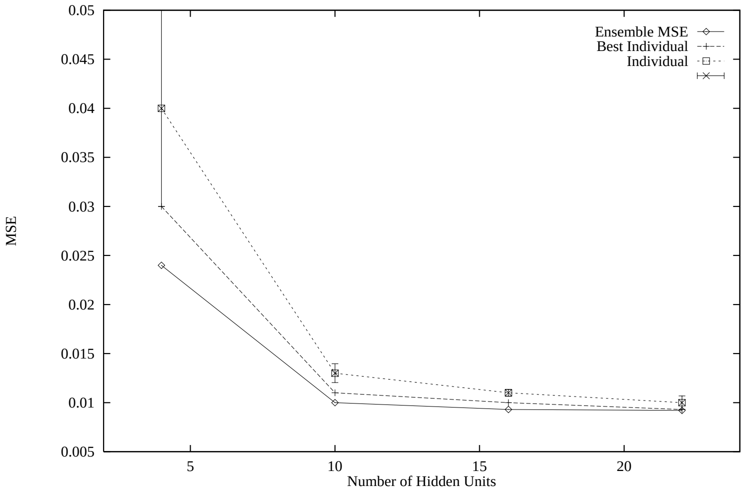


Figure 4.11: NIST Numeral Data: MSE vs. Network Size

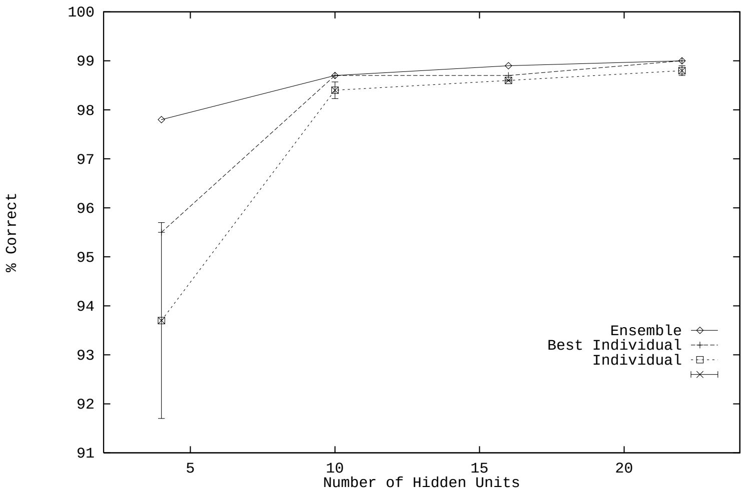


Figure 4.12: NIST Numeral Data: Percent Correct vs. Network Size

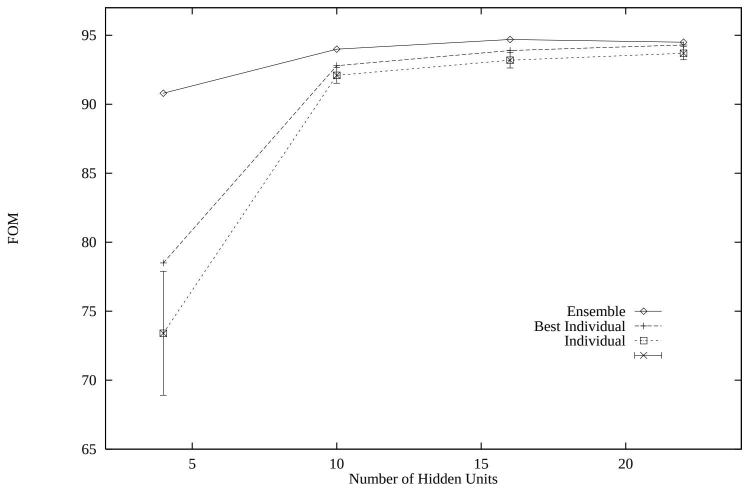


Figure 4.13: NIST Numeral Data: FOM vs. Network Size

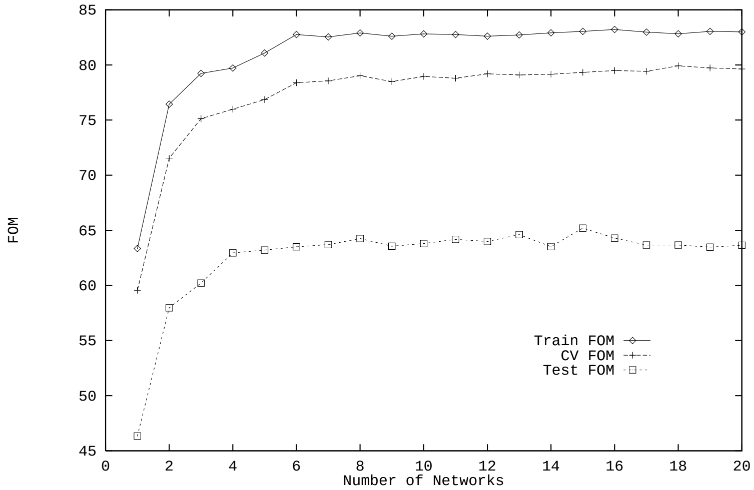


Figure 4.14: Ensemble FOM versus the number of nets in the ensemble. Ensemble FOM graphs for the uppercase training, cross-validatory and testing data sets are shown. Each net in the populations had 10 hidden units. The graphs are for a single randomly chosen ordering of 20 previously trained nets. No effort was made to optimally choose the order in which the nets were added to the ensemble. Improved ordering gives improved results.

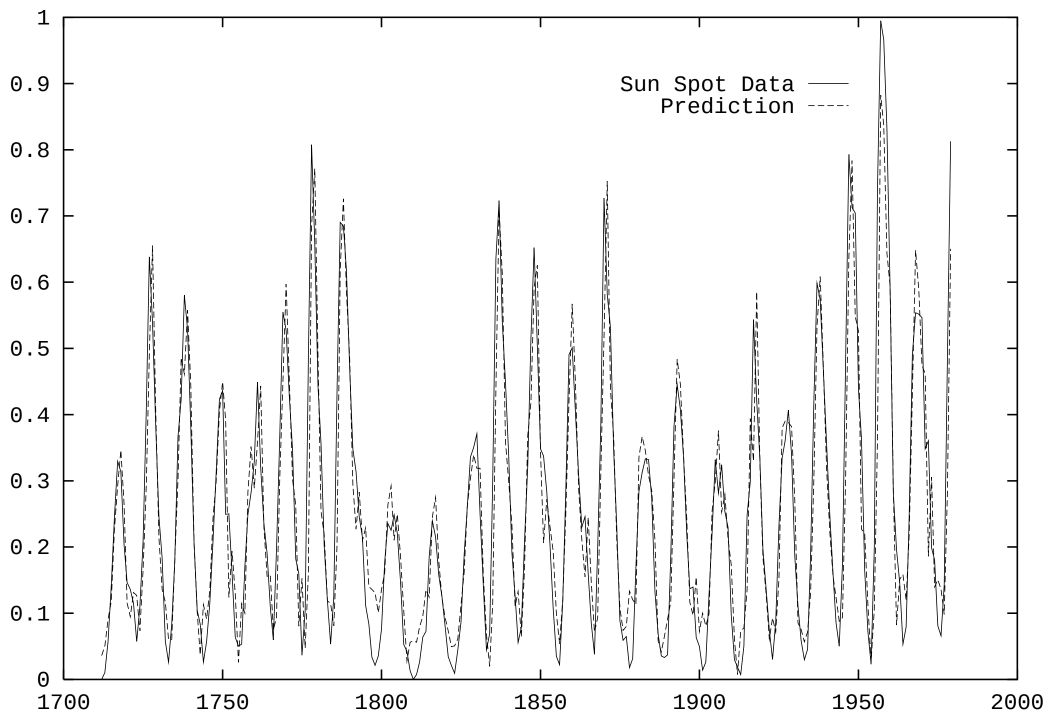


Figure 4.15: Average sunspot activity from 1712 to 1979. Both the real data and the ensemble network prediction are shown.

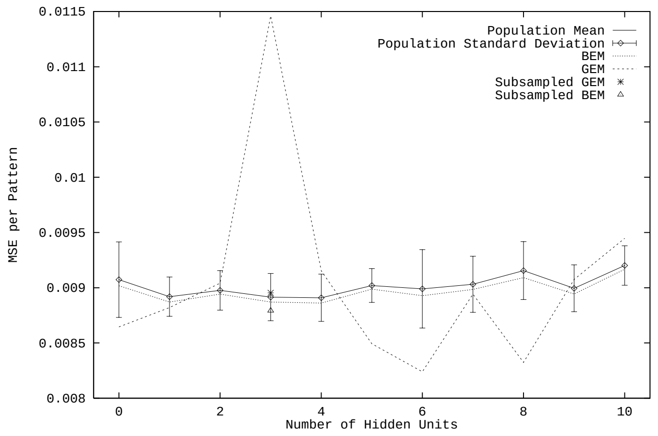


Figure 4.16: MSE of networks trained to perform sunspot activity prediction. (See text for discussion.)

Chapter 5

Application to Neural Hardware

In this Chapter we discuss the averaging method applied to the current state-of-the-art in neural network VLSI implementations: the Intel Ni1000. We show that by applying the averaging method to the Ni1000, we can perform efficient, high-dimensional backpropagation for multilayer perceptrons in a fraction of the time it takes normal computers.

5.1 The Intel Ni1000 VLSI Chip

The Nestor/Intel radial basis function neural chip (Ni1000) contains the equivalent of 1024 256-dimensional artificial digital neurons with 5 bit resolution as well as 64 outputs capable of graded responses(Sullivan, 1993; Cooper, 1993). The chip includes a 20 Mips RISC CPU that supports learning and other operations. In addition it contains a five-stage, pipelined, fixed floating point mathematics unit. The Ni1000 uses CHMOSIV flash technology that permits unpowered, ten-year data retention. It contains 3.62 million transistors in an area of 13.2x15.2 millimeters. The chip can perform 20 billion integer operations per second and is several hundred times faster than conventional microprocessor given the same number of transistors.

The 1024 neurons with 256 dimensional input can perform 1024x256x5 bit comparisons each two μsec. Thus with full dimensionality, the chip is capable of at least 40,000 classifications per second (25 μsec per classification). On average, it is expected to be able to perform at a speed of about 15 μsec per classification.

In training mode, the weight commitment time is about 2 μsec per prototype. Thus to execute a typical RCE (Reilly et al., 1982) or PRCE (Scofield et al., 1987) prototype entry requires about 2 μsec. It is expected that in a normal run, one could perform 300-500 training passes per second (Sullivan, 1993; Cooper, 1993). A normal recognition or classification with RCE or PRCE would be about 1,000 times faster than a Sun Workstation.

The Ni1000 neurons each have a set of learnable weights corresponding to each of the input dimensions. Each neuron also maintains a learnable threshold value. The neurons calculate the "cityblock" distance (i.e. the l_1-norm) between their stored weights and the current input:

$$\text{neuron activity} = \sum_i |w_i - x_i|$$

where w_i is the neuron's stored weight for the ith input and x_i is the ith input. Thus the Ni1000 is ideally suited to perform both the RCE and PRCE algorithms or any of the other commonly used radial basis function algorithms.

The Ni1000 has two drawbacks: it can not calculate dot products in parallel and any activation function beyond the cityblock distance must be calculated serially in a math coprocessor. In most neural network algorithms, dot products and nonlinear activation functions are both crucial components (e.g. multilayer perceptrons). In high dimensional problems, the dot product will be the bottleneck for the calculations. If the dot product can not be done in parallel there will be no advantage to using the Ni1000. As for the activation functions, if the number of hidden units is large, serial calculation of the activation functions will become a bottleneck. One can therefore question the usefulness of the Ni1000 in performing such tasks.

In this chapter, we address these two problems by showing that we can extend the Ni1000 to many of the standard regression and fitting algorithms by combining the averaging described in Chapter 2 with a Cityblock norm approximation to the Euclidean norm in high dimensional spaces. We also discuss methods for speeding up the calculation of the neuron activation functions or "squashing" functions.

5.2 "Fast" Activation Functions

On way to speed-up the Ni1000 is to use "fast" activation functions.

The two most common activation functions for neural networks are the sigmoid (translated and scaled hyperbolic tangent), $f(x)$, and the standard Gaussian, $g(x)$,

$$f(x) = (1 + e^{-x})^{-1},$$

$$g(x) = e^{-x^2}.$$

Both of these functions have been used extensively in neural network algorithms because their derivatives have very simple forms and can be calculated quickly and used in the backpropagation algorithm. In particular

$$f\prime(x) = f(x)(1 - f(x)),$$

and

$$g\prime(x) = -2xg(x).$$

The problem with these common activation functions is that they require the calculation of a transcendental function which can be more than an order of magnitude longer to compute than a multiplication.

As alternatives we can use (Elliott, 1993; Georgiou and Koutsougeras, 1992; Thrift, 1990)

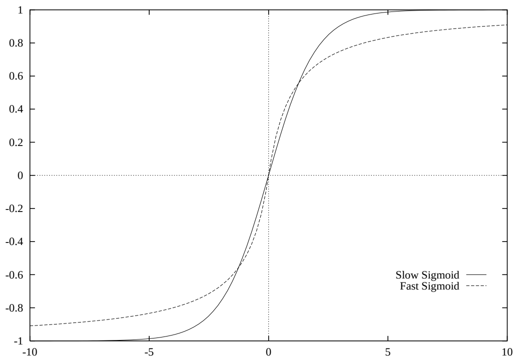
$$\tilde{f}(x) = \frac{x}{1 + |x|},$$

and (Elliott and Perrone, tion; Hanson and Gluck, 1991)

$$\tilde{g}(x) = (1 + x^2)^{-1}$$

which can be used to construct networks which are known to be dense in the space of square integrable functions (Elliott, 1993; Hornik et al., 1989; Duda and Hart, 1973). These activation functions are much faster to compute since they do not require the calculation of a transcendental function. In addition, their derivatives

$$\tilde{f}(x) = (1 - |f(x)|)^2,$$



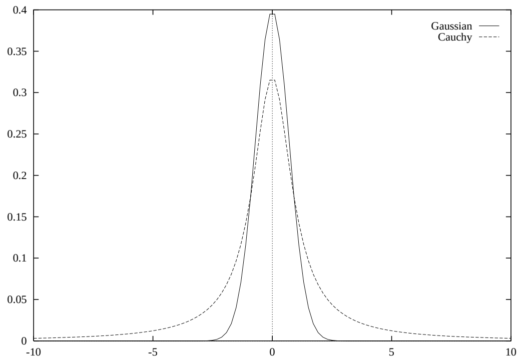


Figure 5.1: A and B. Fig. A shows the sigmoidal activation functions and Fig. B shows the kernel activation function.

DATA SET	HIDDEN UNITS	SLOW ACTIVATION	FAST ACTIVATION
Numbers	22	98.8 ± 0.13	98.7 ± 0.15
Uppercase	30	91.5 ± 0.27	91.4 ± 0.25
Lowercase	30	90.3 ± 0.27	90.1 ± 0.24
Faces	16	95.6 ± 1.9	97.4 ± 0.65

Table 5.1: Comparison of MLP classification performance using the fast and slow activation functions.

and

$$\tilde{g}(x) = -2xg^2(x),$$

also have very simple forms which can be calculated quickly. These functions are plotted in Figure 5.1A and 5.1B; The major qualitative difference between the fast and the slow activation functions is the behavior in the tails: The slow activations approach their asymptotes much more quickly due to their exponential behavior. The differing behavior in the tails may affect overall learning speed (i.e. the number of stochastic iterations before convergence)[1] and even the quality of the solution given a fixed number of hidden units. In order to demonstrate that the fast sigmoid proposed above is a viable alternative to the slow sigmoid, we have performed several simulation using MLP's with identical architectures but different activation functions. Below is a table which compares the performance of MLP networks using $f(x)$ and $\tilde{f}(x)$.[2] In order to accelerate training, the weights of the MLP's trained with $f(x)$ were used as initial weights for the MLP's trained with $\tilde{f}(x)$ for all but the face data which was trained with random initial weights. From this table we can see that the networks have comparable performance.

[1] Convergence in MLP networks using stochastic gradient descent tends to slow down as the network approaches a minimum because data points which have been learned tend to lie in the "saturated" region of the activation function. The derivative in the saturated regions has an exponential drop off and so the adjustment to the weights decreases. For this activation functions with heavy tails, the drop off will be slower and the derivative signal higher.

[2] No simulations were done to compare $g(x)$ and $\tilde{g}(x)$.

5.3 Approximate Dot Products

As stated in Section 5.1, the Ni1000 does not have the ability to calculate dot products in parallel. In this section we discuss a method for overcoming this problem by using an approximation to the dot product which the Ni1000 can calculate in parallel.

Consider the following standard formula for the dot product[3]

$$\vec{x} \cdot \vec{y} = \frac{1}{4}(||\vec{x} + \vec{y}||^2 - ||\vec{x} - \vec{y}||^2).$$

This representation indicates that we do not need to calculate the dot product explicitly if we know the lengths of $\vec{x}, \vec{y} and \vec{x} - \vec{y}$. At first glance, this re-writing of the dot product looks worse than before: now we need to perform 2 dot products to determine the lengths of the sum and difference! In order to avoid this problem and to take advantage of the parallel architecture of the Ni1000, we propose using the following approximation to the lengths of the vectors:

$$\vec{x} \cdot \vec{y} = \frac{1}{4n}(|\vec{x} + \vec{y}|^2 - |\vec{x} - \vec{y}|^2)$$

where n is the dimension of the vectors and $|\cdot|$ is the cityblock length. We have approximated the Euclidean length with the Cityblock length.

The motivation for this approximation comes from the fact that in high dimensional spaces this approximation is quite good for most of the points in the space. Of course, there are always some points in the space for which this approximation will be very bad; but as the dimensionality increases these points occupy less and less of the space. If we consider that our data is being randomly sampled from some n-dimensional space, then the chance that a point will lie in a region where the approximation is poor will decrease as the dimensionality increases. Thus, the approximation is accurate most of the time. For a rigorous mathematical analysis of this approximation, the reader is referred to AppendixB.

In Figure 5.2, we suggest an intuitive interpretation of why this approximation is reasonable. The arc corresponds to all of the vectors in one quadrant with the same Euclidean length. The inner line (connecting the two ends of the arc) corresponds to all of the vectors in the quadrant whose cityblock length equals the Euclidean length of the vectors in the arc. The outline line (tangent to the arc) corresponds to the set of vectors over which we will be making our approximation. In order to scale the outer line to the inner line, we have to multiple by $1/\sqrt{n}$. The outer line approximates the arc in the region near the tangent point. In high dimensional spaces, this tangent region occupies a large region of the total arc and thus the cityblock distance is a good approximation along most of the arc.

It is clear from the figure that the approximation is only reasonable for about 20% of the points on the arc in 2 dimensions; however in high dimensions non-intuitive things happen.[4] It is possible to show that the approximation, which is quite bad in 2 dimensions, is surprisingly good in high dimensional spaces.

---

[3]Note also that depending on the information available to us, we could use either

$$\vec{x} \cdot \vec{y} = \frac{1}{2}(||\vec{x} + \vec{y}||^2 - ||\vec{x}||^2 - ||\vec{y}||^2)$$

or

$$\vec{x} \cdot \vec{y} = \frac{1}{2}(||\vec{x}||^2 + ||\vec{y}||^2 - ||\vec{x} - \vec{y}||^2).$$

[4]For example, the volume of an n-dimensional unit sphere approaches zero as n increases (See Appendix C.); thus, for a n-dimensional Gaussian probability distribution with $\sigma = 1$, the probability of finding a point in the center of the distribution approaches zero. This is very counter intuitive: In low dimensions, a Gaussian looks like a bump but in high dimensions, the bump is empty! All the data has moved to the tails of the distribution.

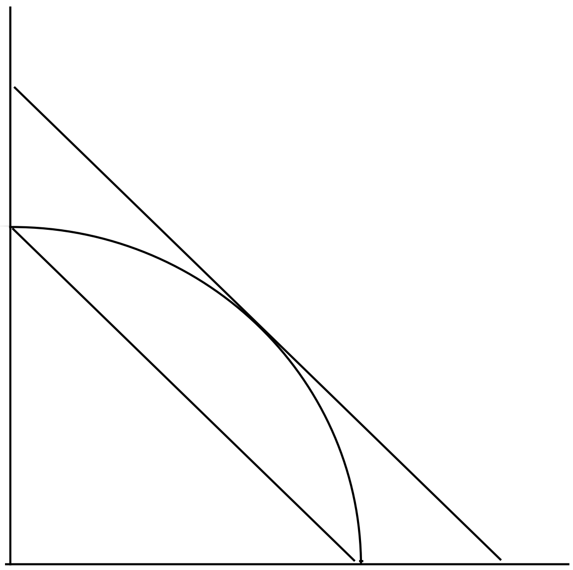


Figure 5.2: Low dimensional interpretation of the cityblock approximation. See the text for details.

In particular, Appendix B analyses this approximation in detail to show that assuming all the vectors of the space are equally likely, the following equation holds:

$$0 < \sigma_n^2 < \left(\frac{2n}{\alpha_n^2(n+1)} - 1\right)\mu_{\text{lower}}^2,$$

where μ_{lower} is the lower bound for μ_n and is given by $\mu_{\text{lower}} \equiv \alpha_n S/\sqrt{n}$ and α_n is defined by

$$\alpha_n \equiv \frac{n-1}{n+1}\sqrt{1+\frac{en}{2\pi(n-1)}\left(\frac{n}{2}\right)^{\frac{1}{2n-2}}} + \frac{2}{n+1}.$$

From this equation it is clear that the error in our approximation decreases arbitrarily as the dimension increases. This fact is very good news for many real-world pattern recognition task which can typically have thousand or even tens of thousands of dimensions[5].

Appendix B also shows that in the case in which each dimension of the vector is constrained such that the entire vector can not lie along a single axis

$$\sigma_n^2 \approx \frac{2(n-1)}{(n+1)^2}\left(\sqrt{\frac{n}{S}} - 1\right)^2 \mu_{\text{min}}^2,$$

where S is the cityblock length of the vector in question. This estimate is useful when S is some non-negligible percent of n.

For very high dimensional neural network tasks such as pattern recognition from image data and speech recognition tasks where the dimensionality can easily reach the tens of thousands, the approximation outlined above can be combined with the Ni1000 chip to perform an approximation to the MLP. Thus the Ni1000 can be made to implement the common MLP algorithm while maintaining parallel processing speeds.

[5] As in the case of speech recognition and high resolution image recognition.

5.4 Experimental Results

In Appendix B, we show that the Cityblock approximation to the dot product improves as the dimensionality increases. From these calculations, it is not obvious that the dimensionality of real-world problems is sufficiently high to provide an adequate approximation.

In order to test the performance of the approximation described in Section 5.3, we simulated the behavior of the Ni1000 on a SPARC station in serial. We used the approximation only on the first layer of weights (i.e. those connecting the inputs to the hidden units) where the dimensionality is highest and the approximation is most accurate. The approximation was not used in the second layer of weights (i.e. those connecting the hidden units to the output units were calculated in serial. The reason for using the real dot product in the second layer is that the number of hidden units in our simulations is quite small (on the order of 10); therefore since the dimensionality of the hidden unit space is low, the approximation is not accurate. In practice, if the number of hidden units is sufficiently large, the approximation to the dot product may also be used in the second weight layer. Certainly, using the dot product in the second layer may slow the calculation;[6] however it should be noted that for a 2 layer MLP in which the number of hidden units and output units are much lower than the input dimensionality, the majority of the computation is in the calculation of the dot products in the first weight layer. So even using the approximation only in the first layer will significantly accelerate the calculation.

In the simulations, the networks used the approximation when calculating the dot product only in the feedforward phase of the algorithm. For the feedbackward phase (i.e. the error backpropagation phase), the algorithm was identical to the original backward propagation algorithm. In other words the approximation was used to calculate the network activity but the stochastic gradient term was calculated as if the network activity was generated with the real dot product. This simplification does not slow the calculation because all the terms needed for the backpropagation phase are calculated in the forward propagation phase In addition, it allows us to avoid altering the backpropagation algorithm to incorporate the derivative of the cityblock approximation.

In practice, the price we pay for making the approximation is reduced performance. We can ameliorate this problem to a certain extent by increasing the number of hidden units and thereby allow more flexibility in the network. This increase in size will not significantly slow the algorithm since the hidden unit activities are calculated in parallel. [7] Unfortunately, this is only a partial solution to the decrease in performance because, using the approximation, we lose our guarantee that our stochastic gradient descent process will lead us to the optimal solution or even a local minimum. If decreased performance is unacceptable, the learning can still be accelerated by using the approximate dot product until we are near a local minimum and then switch over to the exact dot product for "fine-tuning". This approach will accelerate training but will leave on-line testing at the serial processing rate.

In Table 5.2 and Table 5.3, we compare the performance of a standard MLP without the Cityblock approximation to a MLP using the Cityblock approximation to calculate network activity. In all cases, a population of 10 neural networks were trained from random initial weight configurations and the means and standard deviations were listed. The number of hidden units was chosen to give a reasonable size network while at the same time reasonably quick training. Two observations should be made from these data: 1) The networks using the

[6] For the Ni1000 this is not necessarily true since the on-chip math coprocessor can perform a low-dimensional, second layer dot product while the high-dimensional, first layer dot product is being approximated in parallel by the cityblock units.

[7] In the simulations presented here, everything was computed in serial; so in order to save time, the number of hidden units was not increased as is suggested in the text.

DATA SET	HIDDEN UNITS	STANDARD % CORRECT	CITYBLOCK % CORRECT	BEM CITYBLOCK
Faces	12	94.6±1.4	92.2±1.9	96.3
Numbers	10	98.4±0.17	97.3±0.26	98.3
Lowercase	20	88.9±0.31	84.0±0.48	88.6
Uppercase	20	90.5±0.39	85.6±0.89	90.7

Table 5.2: Comparison of MLPs classification performance with and with out the Cityblock approximation to the dot product. The final column shows the effect of function space averaging.

DATA SET	HIDDEN UNITS	STANDARD FOM	CITYBLOCK FOM	BEM CITYBLOCK
Numbers	10	92.1±0.57	87.4±0.83	92.5
Lowercase	20	59.7±1.7	44.4±2.0	62.7
Uppercase	20	60.0±1.8	44.6±4.5	66.4

Table 5.3: Comparison of MLPs FOM with and with out the Cityblock approximation to the dot product. The final column shows the effect of function space averaging.

approximation do not perform as well as the networks using the real dot product which was expected. 2) The relative performance of the approximating networks is not bad!

Of course, we can now employ the averaging method described in Section 2.2 to further improve the performance of the approximate networks. These results are given in the last column of the table. From these data we see that by combining the cityblock approximation with the averaging method, we can generate networks which have comparable and sometimes better performance than the standard MLPs! In addition, because the Ni1000 is running in parallel, there is minimal additional computational overhead for using the averaging.[8]

Thus we have shown that we can generate an approximate MLP on the Ni1000 which has comparable performance to a standard MLP but which has the advantages that it is much faster and can have many more hidden units than its serial counterpart with minimal overhead. These results are very promising. They illustrate that it is possible to use the inherent high dimensionality of real-world problems to our advantage. Much more work needs to be done in this area to develop a viable version of the MLP for the Ni1000. For example, it is possible to completely replace the dot product with the Cityblock approximation in both the feedforward and error backpropagation phases. This replacement in the backpropagation phase would allow us to calculate an exact derivative for the network and thereby guarantee that our stochastic

[8] The averaging can also be applied to the standard MLPs with a corresponding improvement in performance. However, for serial machines averaging slows calculations by a factor equal to the number of averaging nets. In order to avoid this slow down, one must turn to a parallel processor - like the Ni1000! So we have a Catch-22.

gradient descent would converge to a local minimum. This convergence property should improve network performance. This approach has not been attempted. An addition, there is currently no proof that such a network would be dense in the set of continuous functions. This type of proof would be necessary to put a "Cityblock" MLP on a firm theoretical foundation and to win acceptance from other researchers in the field.[9] Work in this area is continuing.

[9] One can of course argue that if it works, it doesn't matter who accepts it; but I suspect that that view is rather dangerous!

Chapter 6

Hybrid Algorithms

The averaging methods described in Chapter 2 are perhaps the simplest forms of hybrid algorithm. This chapter examines an extension to the notion of averaging which uses local information to select weights for the averaging process. This approach allows the weights to vary as a function of the input. This flexibility takes advantage of the fact that in any given region of input space, some networks may be more reliable than others. This variability gives rise to the notion of a "local expert" or a network which is specialized to a particular region of input space and has been used by other researchers to improve network performance (Reilly et al., 1988; Jacobs et al., 1991).

6.1 Winner-Take-All Confidence Controller

A natural alternative to the averaging methods is to perform classification based on the network with the highest output from a population of networks. This method is known as "winner-take-all" classification (Touretzky, 1989). It has been suggested (Jacobs et al., 1991) that this kind of "hard competition" between the members of the population is not optimal. Therefore instead of just using the network activities to choose a winner, we have examined a hybrid algorithm which uses a winner-take-all (WTA) controller on the confidence (Section 4.4) of the individual networks to decide which network is allowed to classify the input. We used the classification confidence described in Section 4.4. These results are presented in Table 6.1. It is clear from these results that the winner-take-all approach is inferior to averaging.

A modification of the winner-take-all approach (Majani et al., 1989) in which we choose K winners from a population and average, could also be implemented. The K-WTA estimator will approach the BEM estimator as K increases to the size of the population. For the K-WTA hybrid, we are faced with the problem of choosing K. This choice could be made with cross-validation but we take a different approach. If we assume that we know nothing about the correct choice of K, then a non-informative prior would be to assume that each value if K is equally likely. We can now use averaging over this prior to find that the overall weighting for the ith network is given by

$$w_i = \frac{2i}{n(n+1)}$$

where n is the number of estimators in the population.

This modification to the K-WTA algorithm now looks like an approximation to the GEM estimator. The crucial difference is that the GEM weights are constants while the modified K-WTA network weights are functions of the feature space and will vary with the input patterns.

DATA SET	HIDDEN UNITS	BEM FOM	WTA FOM
Numerals	4	90.6	80.3
Numerals	10	93.8	93.2
Numerals	16	94.6	93.6
Numerals	22	94.5	93.5
Lowercase	10	60.3	46.5
Lowercase	20	68.4	62.8
Lowercase	30	68.9	62.7
Lowercase	40	70.7	65.4
Uppercase	10	63.8	51.5
Uppercase	20	67.5	60.4
Uppercase	30	67.1	63.5
Uppercase	40	60.0	64.9

Table 6.1: Comparison of BEM and WTA hybrid estimators' test FOM for the NIST Data

These adaptive weights allow the modified K-WTA estimator to avoid one of the basic problems of GEM estimators: In local regions of feature space, the optimal regressor weighting may be quite different from the global weighting given by GEM. Of course, the modified K-WTA network has the disadvantage that the weights are never guaranteed to be optimal.

Another difference difference between the two methods is that the GEM estimator calculates a different weight for each output dimension of each network. The modified K-WTA network implemented here has a single weight for all of its inputs because we have used the confidence measure which depends in principle on all of the inputs. If we abandon the confidence measure and simply use the network output to rank the nets then each output of each network could have its own weight. Note that this procedure is equivalent to a prior biased towards high network activity whereas the modified K-WTA described above has a bias towards high confidence.

Tables 6.2 and 6.3 compare the performance of the modified K-WTA estimators with the BEM estimators on the test data sets. It is clear that the K-WTA network is reducing the MSE of the estimators as compared to the BEM estimators. The FOM does not reflect this performance improvement. Note however that the best FOM is for a K-WTA network. These facts suggest that the K-WTA is a promising approach.

DATA SET	HIDDEN UNITS	BEM MSE	KWTA MSE	BEM FOM	KWTA FOM
Numbers	4	0.0237	0.0181	90.6	91.0
Numbers	10	0.0104	0.00908	93.8	93.8
Numbers	16	0.00927	0.00828	94.6	94.3
Numbers	22	0.00917	0.00816	94.5	94.3
Uppercase	10	0.0263	0.0236	63.8	63.3
Uppercase	20	0.0211	0.0196	67.5	67.3
Uppercase	30	0.0195	0.0184	67.1	67.2
Uppercase	40	0.0192	0.0182	68.1	69.0
Lowercase	10	0.0314	0.0282	60.3	59.7
Lowercase	20	0.0232	0.0212	68.4	67.7
Lowercase	30	0.0213	0.0196	68.9	68.7
Lowercase	40	0.0205	0.0190	70.7	71.1

Table 6.2: Comparison of the test performance of the BEM and K-WTA estimators.

DATA SET	HIDDEN UNITS	BEM MSE	KWTA MSE	BEM % CORRECT	KWTA % CORRECT
Faces	4	0.0602	0.0444	97.5	95.0
Faces	8	0.0265	0.0188	96.2	96.9
Faces	12	0.0192	0.0141	97.5	97.5
Faces	16	0.0189	0.0146	98.1	98.1

Table 6.3: Comparison of the test performance of the BEM and K-WTA estimators.

Appendix A

Related Statistical Results

It is known that neural networks are dense in the set of square integrable functions (Hornik et al., 1989; Hornik et al., 1990; Hornik, 1991; Cybenko, 1989; Funahashi, 1989). However, simply knowing that a desired function exists in the space of possible neural network functions does not guarantee that our neural network algorithms will converge to the desired function. How does one know that the function which optimizes the standard neural network costs (e.g. the MISE and the MLE) corresponds to the desired function? In this appendix, we prove that the optimizers do indeed correspond to the desired solutions.

A.1 MSE as an Estimate of MISE

For finite data, neural network algorithms typically minimize the empirical mean square error,

$$\widehat{\mathrm{MSE}}[f] = \frac{1}{N}\sum_i (f(x_i) - y_i)^2$$

where N is the number of data points (x, y). In the large N limit, the Law of Large Numbers implies that

$$\lim_{N\to\infty} \frac{1}{N}\sum_i (f(x_i) - y_i)^2 = \int E[(f(x) - y)^2]dx$$

or in other words that the empirical mean square error converges to the mean integrated square error in probability, i.e.

$$\lim_{N\to\infty} \widehat{\mathrm{MSE}}[f] = \mathrm{MISE}[f].$$

Thus as the size of our data increases, our approximation of the optimal function converges to the correct solution.

A.2 MSE and Classification

In this section, it is shown that the function which minimizes the mean square error in classification problems is also the probability density function.

Consider the following non-deterministic classification problem: Define a class $\mathcal{C}$ and a discrete random variable $t(x)$ such that $t(x) = 1$ when $x \in \mathcal{C}$ and $t(x) = 0$ when $x \in \overline{\mathcal{C}}$. Thus

$$t(x) = \begin{cases} 1, \text{with probability } p(t = 1|x) \\ 0, \text{with probability } p(t = 0|x) \end{cases}$$

where $p(t|x)$ is the probability density of class $\mathcal{C}$. We want to define a function $f(x)$ which will be a predictor for $t(x)$. The mean integrated square error of $f(x)$ is given by

$$\text{MISE}[f] = \int \Big[(1 - f(x))^2 p(t = 1|x) + f^2(x) p(t = 0|x)\Big] dx.$$

We can find the $f(x)$ which minimizes this cost by minimizing $\text{MISE}[f + \epsilon\phi]$ with respect to ϵ for arbitrary $\phi(x)$. Thus

$$\partial_\epsilon \text{MISE}[f + \epsilon\phi]\Big|_{\epsilon=0} = 0$$

which implies that

$$\int \Big[(1 - f(x)) p(t = 1|x) + f(x)(1 - p(t = 1|x))\Big] \phi(x) dx = 0.$$

Since this equation must be true for all $\phi(x)$, we have

$$(1 - f(x)) p(t = 1|x) + f(x)(1 - p(t = 1|x)) = 0,$$

and therefore the optimal $f(x)$ satisfies

$$f(x) = p(t = 1|x).$$

Thus a neural network or any other classification scheme which attempts to minimize the mean square error is approximating the *a posteriori* probabilities.

Also note that in the case of $\{0, 1\}$-classification, the expected value of the label variable given x is the *a posteriori* probability, i.e.

$$E[t|x] = p(t = 1|x).$$

This demonstrates that classification is simply a special case of regression.

A.3 MSE and Regression

In this section, it is shown that the function, $f(x)$, which minimizes the mean square error in regression problems is the expected value of the random variable given x.

Consider the standard regression problem: Define a random variable y such that

$$y = E[y|x] + n(x)$$

where $n(x)$ is zero mean, uncorrelated noise at x. We want to define a function $f(x)$ which will be a predictor for y. The mean square error of $f(x)$ is given by

$$\begin{aligned}
\text{MISE}[f] &= \int E\Big[(f(x) - y)^2\Big] dx \\
&= \int E\Big[(f(x) - E[y|x])^2\Big] dx + \\
&\quad \int E\Big[2(f(x) - E[y|x])(E[y|x] - y)\Big] dx + \\
&\quad \int E\Big[(E[y|x] - y)^2\Big] dx
\end{aligned}$$

$$
\begin{aligned}
&= \int (f(x) - E[y|x])^2 dx + \\
&\quad \int (f(x) - E[y|x]) E[n(x)] dx + \\
&\quad \int E[n^2(x)] dx \\
&= \int (f(x) - E[y|x])^2 dx + \int E[n^2(x)] dx
\end{aligned}
$$

Since $f(x)$ does not appear in the second term of the final equality above, the $f(x)$ which minimizes the MISE$[f]$ is given by

$$f(x) = E[y|x].$$

A.4 Equivalence of MSE and MLE

In this section it is shown that, under the assumption of Gaussian noise, the MLE optimization process is identical to the MSE optimization process.

Consider the standard regression problem: Define a random variable y such that

$$y = E[y|x] + n(x)$$

where $n(x)$ is zero mean, uncorrelated noise at x. The likelihood function for data $\mathcal{D} = \{(x_i, y_i)\}$ is given by

$$L(\mathcal{D}|f) = \prod_i p(n_i|x_i, f)$$

where $p(\cdot)$ is the probability of observing a certain noise given x and f; and f is the underlying model which is assumed to have generated the data. Our goal is to find the most likely model for the process which generated the data (i.e. maximize the likelihood). If we assume the the structure of the noise is Gaussian, then we have that

$$L(\mathcal{D}|f) = \prod_i \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(f(x_i)-y_i)^2}{2\sigma^2}}$$

since $n(x) = y - f(x)$ under the assumption that $f(x)$ is the true model. Now we note that maximizing the likelihood function is equivalent to minimizing the negative of the log-likelihood function. The negative log-likelihood function is given by

$$
\begin{aligned}
-\ln L(\mathcal{D}|f) &= \frac{1}{\sigma^2} \sum_i (f(x_i) - y_i)^2 + N \ln(2\pi\sigma^2) \\
&= \frac{N}{\sigma^2} \mathrm{MISE}[f] + N \ln(2\pi\sigma^2)
\end{aligned}
$$

where N is the number of data points. From this relationship we see that the likelihood is maximized when the mean square error is minimized.

A.5 MLE and Density Estimation

In this section, it is shown that the function which maximizes the log-likelihood function is also the probability density function.

Without loss of generality, the log-likelihood may be written as

$$\ln L(\mathcal{D}) = \frac{1}{N}\sum_i \ln p(n_i|x_i)$$

where $p(n|x)$ is the probability of observing a noise n at point x. In the limit as $N \to \infty$, the Law of Large Numbers implies that

$$\lim_{N\to\infty} \ln L(\mathcal{D}|f) = \int\int q(n|x)\ln p(n|x)dndx$$

where q(n—x) is the true probability distribution of the noise given x. We now look for the $p(n|x)$ which maximizes the log-likelihood in the infinite data limit with the constraint that $p(n|x)$ be normalized to 1. If we add the normalization constraint, we have that

$$\mathcal{L}(p) = \int\int \Big[q(n|x)\ln p(n|x) + \lambda(p(n|x)-1)\Big]dndx$$

where λ is a Lagrange multiplier. We can find the $p(n|x)$ which maximizes $\mathcal{L}(p)$ by maximizing $\mathcal{L}(p+\epsilon\phi)$ with respect to ϵ for arbitrary $\phi(n,x)$. Thus

$$\partial_\epsilon \mathcal{L}(p+\epsilon\phi)\Big|_{\epsilon=0} = \int\Big(\frac{q(n|x)}{p(n|x)} + \lambda\Big)\phi(n,x)dndx = 0.$$

Since this equation must be true for all $\phi(n,x)$, we have that

$$q(n|x) = -\lambda p(n|x).$$

Imposing our normalization constraint tells us that the optimal $p(n|x)$ satisfies

$$p(n|x) = q(n|x).$$

Thus a neural network or any other optimization scheme which attempts to maximize the likelihood function is approximating the true distribution which generated the data.

Appendix B

Approximating the l_2-Norm

B.1 Mathematical Analysis

B.1.1 Introduction

For a given city block length there is a continuous range of Euclidean lengths possible. It is easy to see that the Euclidean length, l_n, of any n-dimensional vector with a fixed city block length, S, is bounded by

$$\frac{S}{\sqrt{n}} \leq l_n \leq S,$$

where the lower bound is achieved when all the elements of the vector are equal and the upper bound is achieved when all but one element of the vector are zero. For high dimensional spaces, this bound is not helpful. Even in two dimensions, this bound is not very useful. However, in this appendix we show that assuming uniform distribution of the vectors, the average behavior of the Euclidean length is tightly peaked in a narrow range. Specifically, we will show that

$$0 < \sigma_n^2 < \frac{n-1}{n(n+1)}S^2,$$

and

$$\frac{1}{\sqrt{n}}S < \mu_n < \sqrt{\frac{2}{n+1}}S,$$

where μ_n and σ_n^2 are the mean and variance, respectively, of the n-dimensional Euclidean length given a fixed city block length, S. The bounds on the variance tells us that μ_n is a good approximation for l_n for large n. The bounds on the mean tells us that a reasonable approximation for μ_n is given by

$$\mu_n \approx \sqrt{\frac{1}{2n}}S.$$

Using the above approximation for μ_n, we see that the standard deviation of the Euclidean length is of the same order as the the mean which indicates that using the scaled city block length to approximate the Euclidean distance will frequently lead to large errors. However if the true mean is near its maximum bound, the standard deviation is small in comparison and the scaled city block length will rarely lead to large errors. It is therefore important to develop a more precise bound for the mean.

In addition, it should be noted that although we calculate a mean and variance, the variables in this appendix are not random variables. For a given vector, the Euclidean and Cityblock lengths are fixed not random. This appendix calculates is the amount the true value varies about our estimate.

B.1.2 Definitions

The city block length, S, of an n-dimensional vector is defined as the sum of the absolute values of the elements of the vector:

$$S \equiv \sum_{i=1}^{i=n} |x_i|.$$

The Euclidean length, l_n, of an n-dimensional vector is defined as the square root of the sum of the squares of the elements of the vector:

$$l_n \equiv \sqrt{\sum_{i=1}^{i=n} x_i^2}.$$

The mean, μ_n, of the Euclidean length of an n-dimensional vector with fixed city block length, S, is the expected value of the length over all possible lengths:

$$\mu_n \equiv E_S[l_n].$$

The variance, σ_n^2, of the Euclidean length of an n-dimensional vector with fixed city block length, S, is the expected value of the square of the difference between the length and the mean length:

$$\sigma_n^2 \equiv E_S[(l_n - E_S[l_n])^2].$$

B.1.3 Assumptions

The calculations in the appendix are exact assuming that all possible Euclidean lengths for a fixed city block length are equally likely. If a localized group of vectors is more likely, the variance will be reduced. If on the other hand there are two or more disjoint localized groups of vectors which are more likely, the variance may increase.

In addition, the calculations in this appendix are exact assuming that the elements of the vectors can take on any real value. On a computer, however, there is a finite resolution for the elements. This assumption will break down when the finite resolution effects become large. These effects will become large when $\frac{S}{\sqrt{n}}$ is of the same order as the resolution of the computer arithmetic.

B.1.4 Calculating the Second Moment of l_n

Define the n-dimensional normalizing volume, $V_n(S)$, as

$$V_n(S) \equiv \int_0^S \int_0^{S-x_n} \int_0^{S-x_n-x_{n-1}} \cdots \int_0^{S-x_n-x_{n-1}-\cdots-x_3} dx_2 \cdots dx_n.$$

Note that the limits of the integration impose the constraints that $x_i \in [0, S]$ $\forall i = 1 \ldots n$ and $S = \sum_{i=1}^{i=n} x_i$. We are justified in integrating only over the positive values of x_i by the symmetry of the city block metric which implies that the average over this region is equal to the average over the entire set of vectors with city block length S. Define the integral, $I_n(S)$, of the square of the lengths of the vector in volume $V_n(S)$ as

$$I_n(S) \equiv \int_0^S \int_0^{S-x_n} \int_0^{S-x_n-x_{n-1}} \cdots \int_0^{S-x_n-x_{n-1}-\cdots-x_3} l_n^2 dx_2 \cdots dx_n.$$

It is useful to note that using the constraint, $S = \sum_{i=1}^{i=n} x_i$, gives

$$l_n^2 = \sum_{i=2}^{i=n} x_i^2 + (S - \sum_{i=2}^{i=n} x_i)^2.$$

We can now write the second moment of l_n, the expected value of the square of the lengths of the vectors in volume $V_n(S)$, as

$$E_S[l_n^2] = \frac{I_n(S)}{V_n(S)}.$$

For convenience, define $L_n(S)$ as

$$L_n(S) \equiv E_S[l_n^2].$$

Closed Form for $V_n(S)$

If we transform to $y_i = \frac{x_i}{S}$, we can derive that the scaling law for $V_n(S)$ is

$$V_n(S) = \nu_n S^{n-1},$$

where ν_n is a geometrical factor independent of S. From the definition of $V_n(S)$, we can derive the recursion relation

$$V_n(S) = \int_0^S V_{n-1}(S-x)dx.$$

Combining this recursion relation with the scaling law, we have

$$\nu_n S^{n-1} = \int_0^S \nu_{n-1}(S-x)^{n-2}dx = \nu_{n-1}\frac{1}{n-1}S^{n-1},$$

which simplifies to

$$\nu_n = \frac{1}{n-1}\nu_{n-1}.$$

For $n = 1$, we can calculate from the definition that

$$V_2(S) = S.$$

Therefore

$$\nu_2 = 1,$$

and so by induction we can write

$$\nu_n = \frac{1}{(n-1)!}.$$

Combining this result with the scaling law give the closed form solution for the volume integral,

$$V_n(S) = \frac{1}{(n-1)!}S^{n-1}.$$

Closed form for $L_n(S)$

From the definitions of $V_n(S)$, $I_n(S)$ and $L_n(S)$, we can can derive the recursion relation

$$L_n(S) = \frac{\int_0^S V_{n-1}(S-x)L_{n-1}(S-x)dx}{V_n(S)} + \frac{\int_0^S x^2 V_{n-1}(S-x)dx}{V_n(S)},$$

which using the closed form for $V_n(S)$ can be written as

$$\begin{aligned} L_n(S) &= (n-1)S^{1-n}\int_0^S (S-x)^{n-2}L_{n-1}(S-x)dx \\ &+(n-1)S^{1-n}\int_0^S x^2(S-x)^{n-2}dx. \end{aligned}$$

If we again transform to $y_i = \frac{x_i}{S}$, we can derive that the scaling law for $L_n(S)$ is

$$L_n(S) = \lambda_n S^2,$$

where λ_n is a geometrical factor independent of S. Combining the recursion relation with the scaling law, we can write

$$L_n(S) = (n-1)S^{1-n}\int_0^S \lambda_{n-1}(S-x)^n dx + (n-1)S^{1-n}\int_0^S x^2(S-x)^{n-2}dx$$

which simplifies to

$$\lambda_n S^2 = (n-1)S^2\lambda_{n-1}\int_0^1 (1-x)^n dx + (n-1)S^2\int_0^1 x^2(1-x)^{n-2}dx$$

This equation gives the following recursion relation

$$\lambda_n = \frac{(n-1)}{n+1}\lambda_{n-1} + \frac{2}{(n+1)n}.$$

If we now transform to g_n,

$$g_n \equiv (n+1)n\lambda_n,$$

then

$$g_{n-1} = n(n-1)\lambda_{n-1}$$

and the recursion relation for λ can be written as

$$g_n = g_{n-1} + 2.$$

For $n = 2$, we can solve for $L_2(S)$ from the definition which gives

$$L_2(S) = \frac{2}{3}S^2.$$

Therefore

$$\lambda_2 = \frac{2}{3}$$

and so

$$g_2 = 4.$$

Using induction, we can now solve for g_n in closed form,

$$g_n = 2n.$$

So we can write λ_n in closed form as

$$\lambda_n = \frac{2}{n+1}.$$

Combining this equation with the scaling law gives

$$E[l_n^2] = \frac{2}{n+1}S^2.$$

B.1.5 Bounds on the Mean and Variance for Fixed S

From the definition of σ_n^2,

$$\sigma_n^2 \equiv E[(l_n - E[l_n])^2],$$

we can write

$$\sigma_n^2 = E[l_n^2] - E^2[l_n].$$

Since

$$\sigma_n^2 > 0$$

and we have a closed form for $E[l_n^2]$, we can bound the mean above by

$$E[l_n] < \sqrt{\frac{2}{n+1}}S.$$

Combining this upper bound with the naive lower bound,

$$\frac{1}{\sqrt{n}}S \leq l_n,$$

we can write the bounds for the mean as

$$\frac{1}{\sqrt{n}}S < E[l_n] < \sqrt{\frac{2}{n+1}}S.$$

If we negate this equation and add $L_n(S)$ we have

$$\frac{2}{n+1}S^2 - \frac{1}{n}S^2 > E[l_n^2] - E^2[l_n] > 0$$

which is a bound for the variance and can be written as

$$0 < \sigma_n^2 < \frac{n-1}{n(n+1)}S^2.$$

B.1.6 Improving the Lower Bound on the Mean

With the bounds stated above, we can not say with certainty how often our estimates of the Euclidean distance will lead to errors which are beyond some acceptable limits. If the acceptable limits are large, say 50%, then the bounds presented in the preceding sections are adequate; however, if the acceptable errors must be smaller then the variance must be smaller. We can calculate the variance exactly if we know the mean exactly. We have three avenues open to us: We can try to calculate the mean directly (which because of its difficulty has not been done); we can do a Monte Carlo integration (due to time constraints, exact computer solutions of the integrals can be found only for the smallest of n and S); or we can try to find useful lower bounds on the mean of the Euclidean length. The last method is the approach we take in this section.

In this section, the following improved bound is established:

$$0 < \sigma_n^2 < \left(\frac{2n}{\alpha_n^2(n+1)} - 1\right)\mu_{\text{lower}}^2,$$

where μ_{lower} is the lower bound for μ_n and is given by $\mu_{\text{lower}} \equiv \alpha_n S/\sqrt{n}$ for the tighter bound and by $\mu_{\text{lower}} \equiv S/\sqrt{n}$ for the weaker bound and α_n is defined by

$$\alpha_n \equiv \frac{n-1}{n+1}\sqrt{1 + \frac{en}{2\pi(n-1)}}\left(\frac{n}{2}\right)^{\frac{1}{2n-2}} + \frac{2}{n+1}.$$

A comparison of this new bound with the bound derived in the previous sections is plotted in figure B.1. The tighter bound, although not the tightest possible, shows that the variance is not excessively large.

Before moving to the calculations we note an interesting point: The Monte Carlo simulations are very easy to perform for calculating the mean and variance of the city block distance for a given Euclidean distance. This appendix has taken the approach of looking at the statistics of the Euclidean length given a fixed city block length. It may be fruitful to try calculation the statistics of the city block length given a fixed Euclidean length.

To establish a better lower bound on the mean, μ_n, we rewrite the integral for μ_n,

$$\mu_n \equiv \frac{\int_0^S \int_0^{S-x_n} \int_0^{S-x_n-x_{n-1}} \cdots \int_0^{S-x_n-x_{n-1}-\cdots-x_3} l_n dx_2 \cdots dx_n}{V_n(S)},$$

in the following way

$$\mu_n = \frac{\int_\Delta \sqrt{r^2 + \frac{S^2}{n}} r^{n-2} dr d\Omega}{\hat{V}_n(S)},$$

where Δ is the $(n-1)$-dimensional hypersimplex (i.e. the set of all n-dimensional vectors with city block length S); $\hat{V}_n(S)$ is the volume of Δ and is given by

$$\hat{V}_n(S) \equiv \int_\Delta r^{n-2} dr d\Omega;$$

r is the distance from the center of Δ to any point in Δ; $d\Omega$ is the infinitesimal angular volume element in $n-1$ dimensions; and we have used the fact that

$$l_n^2 = \sum_{i=2}^{i=n} x_i^2 + (S - \sum_{i=2}^{i=n} x_i)^2.$$

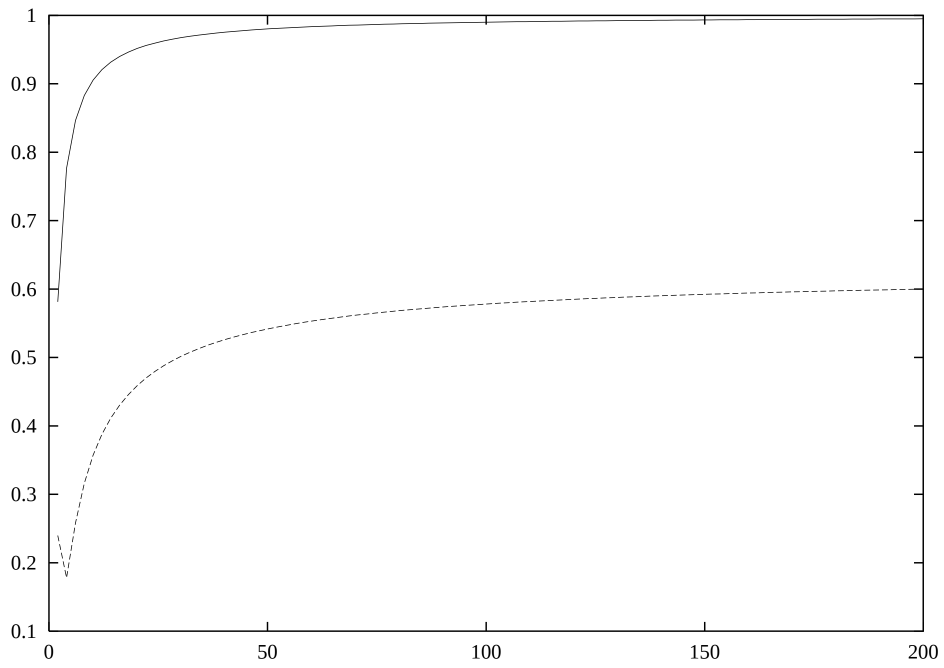


Figure B.1: Plot of $\sigma_n/\mu_{\text{lower}}$ as a function of dimension. The upper line is for the weak lower bound while the lower line is for the tighter lower bound. From the graph we see that for the weak lower bound the standard deviation is the same size as the lower bound; while for the tighter lower bound the standard deviation is about half the lower bound.

If we project the hypersimplex along one dimension, we find that the new volume coincides with the previously defined volume except in one fewer dimensions. By simple geometry, the constant of proportionality between these two volumes is $\sqrt{n}$; so we have that

$$\hat{V}_n(S) = \sqrt{n} V_n(S).$$

Since the integration limits in hyperspherical coordinates are so complex, it is not clear how to perform the integral for μ_n. However, we can bound this integral below with

$$\int_\Omega \int_{r=0}^{r=R_e} \sqrt{r^2 + \frac{S^2}{n}} r^{n-2} dr d\Omega < \int_\Delta \sqrt{r^2 + \frac{S^2}{n}} r^{n-2} dr d\Omega,$$

where Ω is the angular volume of Δ and R_e is defined by

$$C_{n-1} R_e^{n-1} = \hat{V}_n(S),$$

where C_n is the constant of proportionality of an n-dimensional sphere and is given by (see the Appendix C)

$$C_n \equiv \frac{2\pi^{\frac{n}{2}}}{n\Gamma(\frac{n}{2})}.$$

The lower bound integral calculates the average length over a hypersphere of the same volume as the hypersimplex. Since the integral is radially symmetric and monotonic increasing, we are guaranteed that the average over the regions of the hypersphere that lie outside of the hypersimplex must be less than the average over the regions of the hypersimplex that lie outside of the hypersphere; and thus this integral forms a lower bound.

Let

$$M_n(S) \equiv \frac{\int_\Omega \int_{r=0}^{r=R_e} \sqrt{r^2 + \frac{S^2}{n}} r^{n-2} dr d\Omega}{\hat{V}_n(S)}.$$

Since $\int d\Omega = (n-1)C_{n-1}$, we can solve the angular integral immediately which gives

$$M_n(S) = \int_{r=0}^{r=R_e} \sqrt{r^2 + \frac{S^2}{n}} r^{n-2} dr (n-1) \frac{C_{n-1}}{\hat{V}_n(S)}.$$

Solving $C_{n-1} R_e^{n-1} = \hat{V}_n(S)$ for R_e gives

$$R_e = \left(\frac{\hat{V}_n(S)}{C_{n-1}} \right)^{\frac{1}{n-1}},$$

which using Stirling's approximation simplifies to

$$R_e = S \sqrt{\frac{e}{2\pi(n-1)}} \left(\frac{n}{2} \right)^{\frac{1}{2n-2}}.$$

Making a change of variables from r to $x = \frac{\sqrt{n}}{S} r$ gives

$$M_n(S) = \int_{x=0}^{x=\frac{\sqrt{n}}{S} R_e} \sqrt{x^2+1} x^{n-2} dx (n-1) \left(\frac{S}{\sqrt{n}} \right)^n R_e^{1-n}.$$

For $z < 1$, we can bound this integral using

$$\int_{x=0}^{x=z} \sqrt{x^2+1} x^{n-2} dx > \int_{x=0}^{x=z} x^{n-2} + \beta(z) x^n dx,$$

where $\beta(z)$ is defined by

$$\beta(z) \equiv \frac{1}{z^2} (\sqrt{z^2+1} - 1).$$

Integrating gives

$$\mu_n(S) > \alpha_n \frac{S}{\sqrt{n}}.$$

which implies that

$$\sigma_n^2 < \left(\frac{2n}{n+1} - \alpha_n^2 \right) \frac{S^2}{n},$$

where α_n is defined by

$$\alpha_n \equiv \frac{n-1}{n+1} \sqrt{1 + \frac{en}{2\pi(n-1)}} \left(\frac{n}{2} \right)^{\frac{1}{2n-2}} + \frac{2}{n+1}.$$

These calculations provide a tighter lower bound for μ_n and a tighter upper bound for σ_n^2 than the calculations in the preceding sections.

B.1.7 Constraining the Range of Each Vector Element

In this section we consider the case in which each dimension is constrained such that the entire vector can not lie along a single axis. Using these constraints, we show that the variance of the Euclidean distance about the Cityblock distance is approximated by

$$\sigma_n^2 \approx \frac{2(n-1)}{(n+1)^2}\left(\sqrt{\frac{n}{S}}-1\right)^2 \mu_{\min}^2,$$

where S is the Cityblock length of the vector in question. (See figure B.2.) This estimate is useful when S is some non-negligible percent of n since the bound gets tighter like $\frac{1}{n}$ which implies that the approximation, $S \approx \gamma l$, improves with increasing n, where l is the Euclidean length and γ is a constant [1] depending only on the dimension. The constant or proportionality, γ, must be determined.

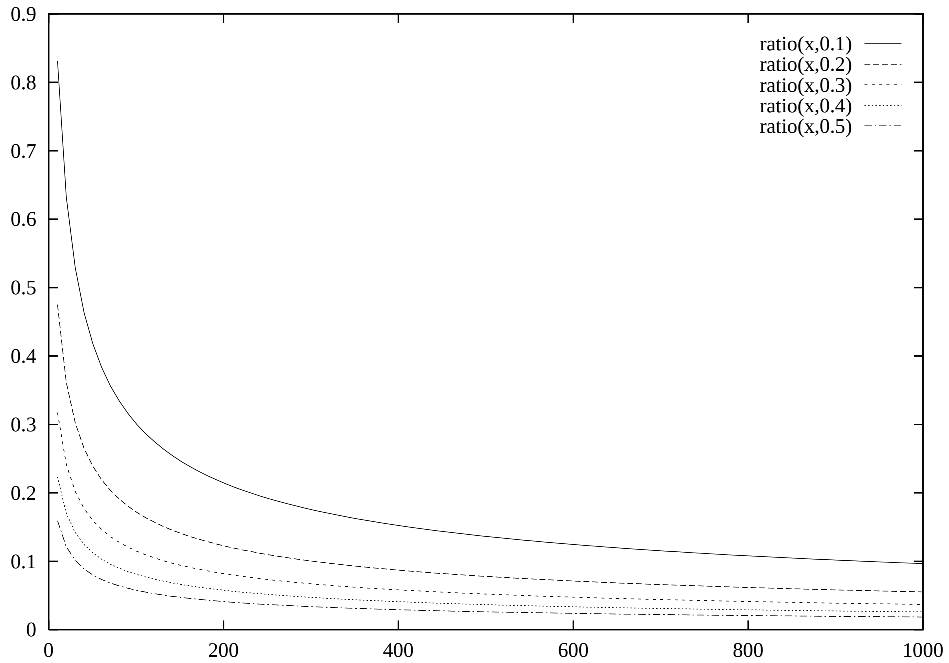


Figure B.2: Plot of $\sigma_n/\mu_{\text{lower}}$ for constrained vectors with varying values of S/n. As S grows the variance shrinks. If we assume that all of the vectors are uniformly distributed in an n-dimensional unit hypercube, it is easy to show that the average Cityblock length is $n/2$ and the variance of the Cityblock length is $n/12$. Since S/n will generally be within one standard deviation of the mean, we find that typically $0.2 < S/n < 0.8$. We can use the same analysis on binary valued vectors to derive similar results. Note that the techniques described in this appendix becomes truly useful in very high dimensions which suggests, for example, that it be used with gray-scale image data.

To derive this approximation of the variance, we constrain each dimension such that $x_i \in$

[1] From their definitions, we see that S and l must scale the same way. Thus S/l must be a constant.

$[0,1]\ \forall i$. [2] Thus, we are demanding that $0 < S < n$. In order to use the formalism of the preceding sections, we would have to perform the integrals over a region in n-dimensional space which can be thought of as a $(n-1)$-dimensional hypersimplex with its corners clipped. Since the clipped hypersimplex has such complex constraining relationships, the formalism of the previous sections breaks down. As an alternative, we choose to find the variance for the hypersphere bounding the clipped hypersimplex and then prove that the variance on the clipped hypersimplex is bounded by this value.

As in the previous sections, the center of the clipped hypersimplex is at the vector $(\frac{S}{n}, \ldots, \frac{S}{n})$. For the unclipped hypersimplex, the vectors farthest from the center have the form

$$(0, \ldots, 0, S, 0, \ldots, 0)$$

and correspond to the corners of the hypersimplex. However, in the case of the clipped hypersimplex, we have constrained the problem such that a single dimension can not contain the entire vector. For the clipped hypersimplex, the vectors which are farthest from the center have S 1's and $n - S$ 0's and correspond to the corners of the clipped hypersimplex. [3] Comparing the center vector to a corner vector, we see that the radius of the bounding hypersphere for the clipped hypersimplex is given by

$$R = \sqrt{\frac{S(n-S)}{n}}.$$

Using this radius, we can calculate the variance in the bounding hypersphere and use this variance as an estimate of the variance in the clipped hypersimplex. Using the same formalism as in section B.1.6 we find that the first moment, $E_S[l_n]$, is estimated by

$$E_S[l_n] \approx \left(1 + \frac{n-1}{n+1}\left(\sqrt{\frac{nR^2}{S^2} + 1} - 1\right)\right)\frac{S}{\sqrt{n}}$$

and the second moment, $E_S[l_n^2]$, is given by

$$E_S[l_n^2] = \left(1 + \frac{n-1}{n+1} \cdot \frac{nR^2}{S^2}\right)\frac{S^2}{n}.$$

Combining these two results, we find that the variance can be estimated by [4]

$$\sigma^2 \approx \frac{n-1}{n+1} \cdot \frac{S^2}{n}\left(\sqrt{\frac{n}{S}} - 1\right)^2 \frac{2}{n+1}.$$

[2] These constraints can be generalized to $x_i \in [0,\alpha]\ \forall i and 0 < \alpha < S$ without changing the results of this section.

[3] For simplicity, we have assumed that S is an integer. If S is non-integer, we can use the smallest integer greater than S to derive the bound on the variance. When S is larger, this approximate bound is quite good. Also note that the elements of the vector are all 0 or 1. These values depend directly on the constraints that we have chosen for the individual dimensions.

[4] Note that we did not use this estimate of the variance in the previous sections because without the constraints on the vector elements, the variance loses its $1/n$ behavior. The $1/n$ behavior is lost because for unconstrained vector elements the bounding hypersphere has $R = S$. It is therefore important that the Cityblock length of the vector be significantly larger than the constraint on each element.

Appendix C

Volume of an n-Dimensional Sphere

In this appendix we calculate the constant of proportionality for an n-dimensional sphere.

The constant of proportionality, C_n, of an n-dimensional sphere is defined by the relation

$$C_n R^n = \int_\Omega \int_0^R r^{n-1} dr d\Omega$$

where $d\Omega$ is the infinitesimal unit of angular volume in n-dimensions. Therefore $C_n = \frac{1}{n}\int d\Omega$. To solve for the angular integral we consider

$$\left[\int_{-\infty}^{+\infty} e^{-x^2} dx\right]^n = \int_\Omega \int_0^\infty e^{-r^2} r^{n-1} dr d\Omega.$$

Thus

$$\pi^{\frac{n}{2}} = \frac{1}{2}\Gamma(\frac{n}{2}) \int_\Omega d\Omega.$$

So we have that

$$C_n = \frac{2\pi^{\frac{n}{2}}}{n\Gamma(\frac{n}{2})}.$$

Bibliography

Baldi, P. (1991). Computing with arrays of bell-shaped and sigmoid functions. In *Advances in Neural Information Processing Systems 3*. Morgan Kaufmann.

Baldi, P. and Chauvin, Y. (1991). Temporal evolution of generalization during learning in linear netwroks. *Neural Computation*, 3:589–603.

Banan, M. and Hjelmstad, K. D. (1992). Self-organization of architecture by simulated heirarchical adaptive random partitioning. In *International Joint Conference on Neural Networks*, pages III:823–828. IEEE.

Barron, A. R. (1991). Complexity regularization with applications to artificial neural networks. In Roussas, G., editor, *Nonparametric Functional Estimation and Related Topics*, pages 561–576. Kluwer.

Baxt, W. G. (1992). Improving the accuracy of an artificial neural network using multiple differently trained networks. *Neural Computation*, 4(5).

Beckenbach, E. F. and Bellman, R. (1965). *Inequalities.* Springer-Verlag.

Breiman, L. (1992). Stacked regression. Technical Report TR-367, Department of Statistics, University of California, Berkeley.

Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees.* Wadsworth and Brooks/Cole Adv. Books and Software, Pacific Grove, CA.

Bridle, J. S. (1990). Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of paramters. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2*, pages 211–217. Morgan Kaufmann.

Bridle, J. S. and Cox, S. J. (1991). RecNorm: simultaneous normalization and classification applied to speech recognition. In *Advances in Neural Information Processing Systems 3.*

Bridle, J. S., Heading, A. J. R., and MacKay, D. J. C. (1992). Unsupervise classifiers, mutual information and 'phantom targets'. In Moody, J. E., Hanson, S. J., and Lippmann, R. P., editors, *Advances in Neural Information Processing Systems 4*, pages 1096–1101. Morkan Kaufmann.

Buntine, W. L. and Weigend, A. S. (1992). Bayesian back-propagation. *Complex Systems*, 5:603–643.

Carroll, R. J. and Ruppert, D. (1988a). *Transformation and Weighting in Regression.* Monographs on Statistics and Applied Probability. Chapman and Hall.

Carroll, R. J. and Ruppert, D. (1988b). *Transformation and weighting in regression.* Chapman and Hall.

Cooper, L. N. (1991). Hybrid neural network architectures: Equilibrium systems that pay attention. In Mammone, R. J. and Zeevi, Y., editors, *Neural Networks: Theory and Applications*, volume 1, pages 81–96. Academic Press.

Cooper, L. N. (1993). Personal communication.

Cottrell, G. W. and Metcalfe, J. (1991). Empath: Face, emotion, and gender recognition using holons. In Lippmann, R. P., Moody, J. E., and Touretzky, D. S., editors, *Advances in Neural Information Processing Systems 3*, pages 564–571. Morgan Kaufmann.

Cybenko, G. (1989). Approximations by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314.

Denker, J. S., Gardner, W. R., Graf, H. P., Henderson, D., Howard, R. E., Hubbard, W., Jackel, L. D., Baird, H. S., and Guyon, I. (1989). Neural network recognizer for hand-written zip code digits. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems*, pages 323–331, San Mateo, CA. Morgan Kaufmann.

Devroye, L. (1987). *A Course in Density Estimation.* Birkhauser.

Drucker, H., Schapire, R., and Simard, P. (1993). Improving performance in neural networks using a boosting algorithm. In Hanson, S. J., Cowan, J. D., and Giles, C. L., editors, *Advances in Neural Information Processing Systems 5*, pages 42–49. Morgan Kaufmann.

Drucker, H., Schapire, R., and Simard, P. ([To appear]). Boosting performance in neural networks. *International Journal of Pattern Recognition and Artificial Intelligence.*

Duda, R. O. and Hart, P. E. (1973). *Pattern Classification and Scene Analysis.* John Wiley, New York.

Efron, B. (1979). Bootstrap methods: Another look at the jackknife. *The Annals of Statisitics*, 7(1):1–26.

Efron, B. (1982). *The Jackknife, the Boostrap and Other Resampling Plans.* SIAM, Philadelphia, PA.

Efron, B. and Stein, C. (1981). The jackknife estimate of variance. *The Annals of Statisitics*, 9(3):586–596.

Elliott, D. L. (1993). A better activation function for artificial neural networks. ISR technical report TR 93-8, Univeristy of Maryland.

Elliott, D. L. and Perrone, M. P. ([In preparation]). Fast activation functions for neural networks.

Finnoff, W., Hergert, F., and Zimmermann, H. G. (1993). Extended regularization methods for nonconvergent model selection. In Janson, S. J., Cowan, J. D., and Giles, C. L., editors, *Advances in Neural Information Processing Systems 5*, pages 228–235. Morgan Kaufmann.

Fontaine, T. and Shastri, L. (1992). Character recognition using a modular spatiotemporal connectionist model. Technical report, Computer and Information Science Department, University of Pennsylvania.

Funahashi, K. (1989). On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:183–192.

Galland, C. C. and Hinton, G. E. (1990). Discovering high order features with mean field modules. In *Advances in Neural Information Processing Systems 2*. Morgan Kaufmann.

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

Georgiou, G. M. and Koutsougeras, C. (1992). Complex domain backpropagation. *IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing*, 39(5):330–334.

Golomb, B. A., Lawrence, D. T., and Sejnowski, T. J. (1991). SexNet: A neural network identifies sex from human faces. In Lippmann, R. P., Moody, J. E., and Touretzky, D. S., editors, *Advances in Neural Information Processing Systems 3*, pages 572–577. Morgan Kaufmann.

Gonin, R. and Money, A. H. (1989). *Nonlinear Lp-Norm Estimation.* Marcel Dekker, Inc.

Gradshteyn, I. S. and Ryzhik, I. M. (1980). *Table of Integrals, Series and Products.* Academic Press, Inc.

Gray, H. L. and Schucany, W. R. (1972). *The Generalized Jackknife Statistic.* Dekker, New York, NY.

Guyon, I., Poujaud, I., Personnaz, L., Dreyfus, G., Denker, J., and Cun, Y. L. (1989). Comparing different neural network architectures for classifying handwritten digits. In *International Joint Conference on Neural Networks*, pages II:127–132. IEEE.

Guyon, I., Vapnik, V., Boser, B., Bottou, L., and Solla, S. A. (1992). Structural risk minimization for character recognition. In Moody, J. E., Steven J, H., and Lippmann, R. P., editors, *Advances in Neural Information Processing Systems 4*, pages 471–479. Morgan Kaufmann.

Hall, P. (1992). *The Bootstrap and Edgeworth Expansion.* Springer Series in Statistics. Springer-Verlag.

Hammersley, J. M. and Handscomb, D. C. (1964). *Monte Carlo Methods.* Methuen and Company, Ltd.

Hansen, L. K., Liisberg, C., and Salamon, P. (1992). Ensemble methods for handwritten digit recognition. In Kung, S. Y., Fallside, F., and Kamm, C. A., editors, *Neural Networks for Signal Processing II: Proceedings of the 1992 IEEE Workshop*, pages 333–342. IEEE.

Hansen, L. K. and Salamon, P. (1990a). Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intellignce*, 12(10):993–1001.

Hansen, L. K. and Salamon, P. (1990b). Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1000.

Hanson, S. J. and Gluck, M. A. (1991). Spherical units as dynamic consequential regions: Implications for attention, competition and categorization. In *Advances in Neural Information Processing Systems 2*. Morgan Kaufmann.

Härdle, W. (1990). *Applied Nonparametric Regression*, volume 19 of *Econometric Society Monographs.* Cambridge University Press, New York.

Härdle, W. (1991). *Smoothing Techniques with Implementation in S.* Springer Series in Statistics. Springer-Verlag, New York, NY.

Hardy, G. H., Littlewood, J. E., and Polya, G. (1952). *Inequalities.* Cambridge University Press.

Hastie, H. J. and Tibshirani, R. J. (1990). *Generalized Additive Models.* Chapman and Hall, New York, NY.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4:251–257.

Hornik, K., Stinchombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366.

Hornik, K., Stinchombe, M., and White, H. (1990). Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, 3:551–560.

Intrator, N. (1993). Combining exploratory projection pursuit and projection pursuit regression with application to neural networks. *Neural Computation*, 5:509–521.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural Computation*, 3(2).

Kalos, M. H. and Whitlock, P. A. (1986). *Monte Carlo Methods: Basics*, volume 1. John Wiley & Sons.

Kapur, J. N. and Kesavan, H. (1992). *Entropy Ooptimization Principles with applications.* Academic Press, Boston.

Keeler, J. D. and Rumelhart, D. E. (1992). A self-organizing integrated segmentationand recognition neural net. In Moody, J. E., Hanson, S. J., and Lippmann, R. P., editors, *Advances in Neural Information Processing Systems 4*, pages 496–503.

Keeler, J. D., Runmelhart, D. E., and Leow, W.-K. (1991). Intagrated segmentation and recognition of hand-printed numerals. In Lippmann, R. P., Moody, J. E., and Touretzky, D. S., editors, *Advances in Neural Information Processing Systems 3*, pages 557–563.

Koistinen, P. and Holmstrom, L. (1992). Kernel regression and backpropagation training with noise. In Moody, J. E., Hanson, S. J., and Lippmann, R. P., editors, *Advances in Neural Information Processing Systems 4*, pages 1033–1039. Morgan Kaufmann.

Kullback, S. and Leibler, R. (1951). On information and sufficiency. *Annals of Statitstics*, 22:79–86.

Lapedes, A. S. and Farber, R. M. (1987). Non-linear signal processing using neural networks: Prediction and system modelling. Technical report LA-UR-87-2662, Los Alamos National Laboratory.

Le Cun, Y., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2*, pages 396–404.

Le Cun, Y., Jackel, L. D., Boser, B., Denker, J. S., Graf, H. P., Guyon, I., Henderson, D., Howard, R. E., and Hubbard, W. (1989). Handwritten digit recognition: Aplications of neural network chips and automatic learning. *IEEE Communications Magazine*, pages 41–46.

Lepage, R. and Billard, L., editors (1992). *Exploring the Limits of Bootstrap*. Wiley Seris in Probability and Mathematical Statistics. John Wiley & Sons.

Lincoln, W. P. and Skrzypek, J. (1990). Synergy of clustering multiple back propagation networks. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2*, pages 650–657, San Mateo, CA. Morgan Kaufmann.

Linkser, R. (1989). An application of the principle of maximum information preservation to linear systems. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems*, pages 186–194. Morgan Kaufmann.

Liu, Y. (1993). Neural network model selection using aymptotic jackknife estimator and cross-validation. In Hanson, S. J., Cowan, J. D., and Giles, C. L., editors, *Advances in Neural Information Processing Systems 5*, pages 599–606. Morgan Kaufmann.

MacKay, D. J. C. (1992). Bayesian interpolation. *Neural Computation*, pages 415–447.

Majani, E., Erlanson, R., and Abu-Mostafa, Y. (1989). On the k-winners-take-all network. In Touretzky, D. S., editor, *Neural Information Processing Systems*. Morgan Kaufmann.

Mandler, E. and Schuermann, J. (1988). Combining the classification results of independent classifiers based on the dempster-schafer theory of evidences. In Gelsema and Kanal, editors, *Pattern Recognition and Artificial Intelligence*, pages 381–393, Amsterdam. Elsevier Science.

Marple, S. L. (1987). *Digital Spectral Analysis with Applications*. Prentice-Hall.

Martin, G. L. and Pittman, J. A. (1990). Handwritten digit recognition with a back-propagation network. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2*, pages 405–414.

Martin, G. L. and Rashid, M. (1992). Recognizing overlapping hand-printed characters by centered-object integrated segmentation and recognition. In Moody, J. E., Hanson, S. J., and Lippmann, R. P., editors, *Advances in Neural Information Processing Systems 4*, pages 504–511.

Matan, O., Burges, C. J. C., Cun, Y. L., and Denker, J. S. (1992). Multi-digit recognition using a space displacement neural network. In Moody, J. E., Hanson, S. J., and Lippmann, R. P., editors, *Advances in Neural Information Processing Systems 4*, pages 488–495.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092.

Mikhailov, G. A. (1992). *Optimization of Weighted Monte Carlo Methods*. Springer Series in Computational Physics. Springer-Verlag.

Miller, R. G. (1974). The jackknife - a review. *Biometrika*, 61(1):1–16.

Montana, D. (1992). A weighted probabilistic neural network. In Moody, J. E., Hanson, S. J., and Lippmann, R. P., editors, *Advances in Neural Information Processing Systems 4*, pages 1110–1117. Morgan Kaufmann.

Moody, J. E. (1989). Fast learning in multi-resolution hierarchies. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems*, pages 29–39, San Mateo, CA. Morgan Kaufmann.

Moody, J. E. and Utans, J. (1992). Principled architecture selection for neural networks: Applications to corporate bond rating prediction. In Moody, J. E., Hanson, S. J., and Lippmann, R. P., editors, *Advances in Neural Information Processing Systems 4*, pages 683–690. Morgan Kaufmann.

Moore, A. W. (1992). Fast, robust adaptive control by learning only forward models. In Moody, J. E., Hanson, S. J., and Lippmann, R. P., editors, *Advances in Neural Information Processing Systems 4*, pages 571–578. Morgan Kaufmann.

Morgan, N. and Bourlard, H. (1990a). Generalization and parameter estimation in feedforward nets: some experiments. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2*, pages 630–637. Morgan Kaufmann.

Morgan, N. and Bourlard, H. (1990b). Generalization and parameter estimation in feedforward nets: Some experiments. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2*, pages 630–637. Morgan Kaufmann.

Nadaraya, E. A. (1964). On estimating regression. *Theory of Probability and Its Applications.*

Neal, R. M. (1992). Bayesian mixture modeling by monte carlo simulation. Technical report CRG-TR-91-2, Univeristy of Toronto.

Neal, R. M. (1993). Bayesian learning via stochastic dynamics. In Hanson, S. J., Cowan, J. D., and Giles, C. L., editors, *Advances in Neural Information Processing Systems 5*, volume 5. Morgan Kaufmann, San Mateo, CA.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, San Mateo, CA.

Pearlmutter, B. A. and Rosenfeld, R. (1991). Chaitin-kolmogorov complexity and generalization in neural networks. In *Advances in Neural Information Processing Systems 3.*

Perrone, M. P. (1991). A novel recursive partitioning criterion. In *Proceedings of the International Joint Conference on Neural Networks*, volume II, page 989. IEEE.

Perrone, M. P. (1992). A soft-competitive splitting rule for adaptive tree-structured neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, volume IV, pages 689–693. IEEE.

Perrone, M. P. and Cooper, L. N. (1993a). Coulomb potential learning. In *The Handbook of Brain Theory and Neural Networks.* MIT Press. [To Appear].

Perrone, M. P. and Cooper, L. N. (1993b). Learning from what's been learned: Supervised learning in multi-neural network systems. In *Proceedings of the World Conference on Neural Networks.* INNS. [To appear].

Perrone, M. P. and Cooper, L. N. (1993c). When networks disagree: Ensemble method for neural networks. In Mammone, R. J., editor, *Neural Networks for Speech and Image processing*. Chapman-Hall. [To Appear].

Perrone, M. P. and Intrator, N. (1992). Unsupervised splitting rules for neural tree classifiers. In *Proceedings of the International Joint Conference on Neural Networks*, volume III, pages 820–825. IEEE.

Poggio, T. and Girosi, F. (1990). Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, 247:978–982.

Priestley, M. B. (1981). *Spectral Analysis and Time Series*. Academic Press.

Priestley, M. B. (1988). *Non-linear and Non-stationary Time Series Analysis*. Academic Press.

Qian, M., Gong, G., and Clark, J. (1991). Relative entropy and learning rules. *Physical Review A*, 43(2):1061–1070.

Reilly, D. L., Cooper, L. N., and Elbaum, C. (1982). A neural model for category learning. *Biological Cybernetics*, 45:35–41.

Reilly, D. L., Scofield, C. L., Cooper, L. N., and Elbaum, C. (1988). Gensep: A multiple neural network learning system with modifiable network topology. In *Abstracts of the First Annual International Neural Network Society Meeting*. INNS.

Reilly, R. L., Scofield, C. L., Elbaum, C., and Cooper, L. N. (1987). Learning system architectures composed of multiple learning modules. In *Proc. IEEE First Int. Conf. on Neural Networks*, volume 2, pages 495–503. IEEE.

Reisfeld, D. and Yeshurun, Y. (1992). Robust detection of facial features by generalized symmetry. In *Proceedings of the 11th International Conference on Pattern Recognition*, The Hague, Netherlands.

Reisfeld, D., Yeshurun, Y., Intrator, N., and Edelman, S. (1992). Efficient recognition of automatically normalized face images follwoing random and epp-based dimensionality reduction. Preprint.

Rissanen, J. (1986). Stochastic complexity and modeling. *Annals of Statistics*, 14(3):1080–1100.

Rumelhart, D. E. (1988). learning and generalization. In *IEEE Conference on Neural Networks*, San Diego.

Rumelhart, D. E., McClelland, J. L., and the PDP Research Group (1986). *Parallel Distributed Processing, Volume 1: Foundations*. MIT Press.

Sankar, A. and Mammone, R. (1991). Neural tree networks. In Mammone, R. and Zeevi, Y., editors, *Neural Networks: Theory and Applications*. Academic Press.

Schafer, G. (1976). *A Mathematical Theory of Evidence*. Princeton University Press.

Schafer, G. and Logan, R. (1987). Implementing dempster's rule for hierarchical evidence. *Artificial Intelligence*, 33:271–298.

Schapire, R. (1990). The strength of weak learnability. *Machine Learning*, 5(2):197–227.

Scofield, C., Kenton, L., and Chang, J.-C. (1991). Multiple neural net architectures for character recognition. In *Proc. Compcon, San Francisco, CA, February 1991*, pages 487–491. IEEE Comp. Soc. Press.

Scofield, C. L., Reilly, D. L., Elbaum, C., and Cooper, L. N. (1987). Pattern class degeneracy in an unrestricted storage density memory. In Anderson, D. Z., editor, *Neural Information Processing Systems*. American Institute of Physics.

Skilling, J., editor (1989). *Maximum Entropy and Bayesian Methods*. Kluwer Academic Publishers.

Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions (with discussion). *Journal of the Royal Statistical Society, Series B*, 36:111–147.

Stone, M. (1977a). Asymptotic for and against cross-validation. *Biometrika*, 64:29–35.

Stone, M. (1977b). Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society, Series B*, 39:44–47.

Sullivan, M. (1993). Intel and nestor deliver second-generation neural network chip to DARPA: Companies launch beta site program. Feb. 12.

Thrift, P. (1990). Neural networks and nonlinear modeling. *TI Technical Journal*, November:16–21.

Tong, H. (1983). *Threshold Models in Non-linear Time Series Analysis*, volume 21 of *Lecture NOtes in Statistics*. Springer-Verlag, New York.

Tong, H. (1990). *Non-linear Time Series: A Dynamical Systems Approach*. Oxford University Press.

Tong, H. and Lim, K. S. (1980). Threshold autoregression, limit cylces and cyclical data. *Journal of the Royal Statistical Society*, 42:245.

Touretzky, D. S. (1989). Analyzing the energy landscapes of distributed winner-take-all networks. In Touretzky, D. S., editor, *Neural Information Processing Systems*. Morgan Kaufmann.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142.

Vinod, H. D. and Ullah, A. (1981). *Recent Advances in Regression Methods*. Marcel Dekker, Inc.

Wahba, G. (1990). *Spline Models for Observational Data*. SIAM, Philadelphia.

Wahba, G. and Wold, S. (1975). A completely automatic French curve: fitting spline functions by cross-validation. *Communications in Statistics, Series A*, 4:1–17.

Weigend, A. S., Rumelhart, D. E., and Huberman, B. A. (1990). Backpropagation, weight-elimination and time series prediction. In *Proceedings of the 1990 Connectionist Models Summer School*, pages 105–116. Morgan Kaufmann.

Werbos, P. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University.

Wilks, S. S. (1962). *Mathematical Statistics.* John Wiley and Sons.

Wolpert, D. H. (1990). Stacked generalization. Technical report LA-UR-90-3460, Complex Systems Group, Los Alamos, NM.

Xu, L., Krzyzak, A., and Suen, C. Y. (1990). Associative switch for combining classifiers. Department of Computer Science TR-X9011, Concordia University, Montreal, Canada.

Xu, L., Krzyzak, A., and Suen, C. Y. (1992). Methods of combining multiple classifiers and their applications to handwriting recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(3):418–435.