Maintaining Population Diversity by Minimizing Mutual Information

Yong Liu
The University of Aizu
Tsuruga, Ikki-machi
Aizu-Wakamatsu, Fukushima 965-8580, Japan
yliu@u-aizu.ac.jp

Xin Yao
School of Computer Science
The University of Birmingham
Edgbaston, Birmingham, U.K.
X.Yao@cs.bham.ac.uk

Abstract

Based on negative correlation learning [1] and evolutionary learning, evolutionary ensembles with negative correlation learning (EENCL) was proposed for learning and designing of neural network ensembles [2]. The idea of EENCL is to regard the population of neural networks as an ensemble, and the evolutionary process as the design of neural network ensembles. EENCL used a fitness sharing based on the covering set. Such fitness sharing did not make accurate measurement on the similarity in the population. In this paper, a fitness sharing scheme based on mutual information is introduced in EENCL to evolve a diverse and cooperative population. The effectiveness of such evolutionary learning approach was tested on two real-world problems.

1 Introduction

Neural network ensembles adopt the divide-and-conquer strategy. Instead of using a single network to solve a task, an neural network ensemble combines a set of neural networks which learn to subdivide the task and thereby solve it more efficiently and elegantly [1]. However, designing neural network ensembles is a very difficult task. It relies heavily on human experts and prior knowledge about the problem. Based on negative correlation learning [3, 1] and evolutionary learning, evolutionary ensembles with negative correlation learning (EENCL) was proposed for learning and designing of neural network ensembles [2]. The idea of EENCL is to regard the population of neural networks as an ensemble, and the evolutionary process as the design of neural network ensembles.

The negative correlation learning and fitness sharing [4, 5] were adopted in EENCL to encourage the formation of species in the population. The idea of negative correlation learning is to encourage different individual networks in the ensemble to learn different parts or aspects of the training data, so that the ensemble can better learn the entire training data. In negative correlation learning, the individual networks are trained simultaneously rather than independently or sequentially. This provides an opportunity for the individual networks to interact with each other and to specialize.

Fitness sharing refers to a class of speciation techniques in evolutionary computation. The fitness sharing used in EENCL was based on the idea of the covering set that consists of the same training patterns correctly classified by the shared individuals. This fitness sharing cannot accurately measure the similarity between two individuals. For example, even two individuals have the same covering set, the outputs of two individuals can be quite different. A more accurate similarity measurement between two neural networks in a population can be defined by the explicit mutual information of output variables extracted by two neural networks. The mutual information between two variables, output F_i of network i and output F_j of network j, is given by

$$I(F_i; F_j) = h(F_i) + h(F_j) - h(F_i, F_j) \qquad (1)$$

where $h(F_i)$ is the differential entropy of F_i, $h(F_j)$ is the differential entropy of F_j, and $h(F_i, F_j)$ is the joint differential entropy of F_i and F_j. The equation shows that joint differential entropy can only have high entropy if the mutual information between two variables is low, while each variable has high individual entropy. That is, the lower mutual information two variables have, the more different they are. By minimizing the mutual information between variables extracted by two neural networks, two neural networks are forced to convey different information about some features of their input.

This paper presents further results on how to evolve a cooperative population of neural networks by minimizing mutual information [6]. Negative correlation learning is firstly analyzed in terms of minimization of mutual information on a regression task. Secondly, a fitness sharing based on mutual information is introduced into EENCL. Through minimization of mutual information, a diverse and cooperative population of neural networks can be evolved by EENCL. The effectiveness of such evolutionary learning approach was tested on two real-world problems.

The rest of this paper is organized as follows: Section 2 explores the connections between the mutual information and the correlation coefficient, and explains how negative correlation learning can be used to minimize mutual information. Section 3 analyzes negative correlation learning via the metrics of mutual information on a regression task. Section 4 describes EENCL for evolving a population of neural networks, and explores the connections between fitness sharing and mutual information. Section 5 presents experimental results on EENCL by minimizing mutual information. Finally, Section 6 concludes with a summary of the paper.

2 Minimizing Mutual Information by Negative Correlation Learning

2.1 Minimization of Mutual Information

Suppose the output F_i of network i and the output F_j of network j are Gaussian random variables. Their variances are σ_i^2 and σ_j^2, respectively. The mutual information between F_i and F_j can be defined by Eq.(1) [7]. The differential entropy $h(F_i)$ and $h(F_j)$ are given by

$$h(F_i) = \frac{1}{2}[1 + \log(2\pi\sigma_i^2)] \tag{2}$$

and

$$h(F_j) = \frac{1}{2}[1 + \log(2\pi\sigma_j^2)] \tag{3}$$

The joint differential entropy $h(F_i, F_j)$ is given by

$$h(F_i, F_j) = 1 + \log(2\pi) + \frac{1}{2}\log|det(\Sigma)| \tag{4}$$

where Σ is the 2-by-2 covariance matrix of F_i and F_j. The determinant of Σ is

$$det(\Sigma) = \sigma_i^2\sigma_i^2(1 - \rho_{ij}^2) \tag{5}$$

where ρ_{ij} is the correlation coefficient of F_i and F_j

$$\rho_{ij} = \frac{E[(F_i - E[F_i])(F_j - E[F_j])]}{\sigma_i^2\sigma_j^2} \tag{6}$$

where E indicates the expectation value. Using the formula of Eq.(5), we get

$$h(F_i, F_j) = 1 + \log(2\pi) + \frac{1}{2}\log[\sigma_i^2\sigma_i^2(1 - \rho_{ij}^2)] \tag{7}$$

By substituting Eqs.(2),(3), and (7) in (1), we get

$$I(F_i; F_j) = -\frac{1}{2}\log(1 - \rho_{ij}^2) \tag{8}$$

From Eq.(8), we may make the following statements:

1. If F_i and F_j are uncorrelated, the correlation coefficient ρ_{ij} is reduced to zero, and the mutual information $I(F_i; F_j)$ becomes very small.

2. If F_i and F_j are highly positively correlated, the correlation coefficient ρ_{ij} is close to 1, and mutual information $I(F_i; F_j)$ becomes very large.

Both theoretical and experimental results [8] have indicated that when individual networks in an ensemble are unbiased, average procedures are most effective in combining them when errors in the individual networks are negatively correlated and moderately effective when the errors are uncorrelated. There is little to be gained from average procedures when the errors are positively correlated. In order to create a population of neural networks that are as uncorrelated as possible, the mutual information between each individual neural network and the rest of population should be minimized. Minimizing the mutual information between each individual neural network and the rest of population is equivalent to minimizing the correlation coefficient between them.

2.2 Negative Correlation Learning

We consider estimating y by forming an neural network ensemble whose output is a simple averaging of outputs F_i of a set of neural networks by given the training data set $D = \{(\mathbf{x}(1), y(1)), \cdots, (\mathbf{x}(N), y(N))\}$. All the individual networks in the ensemble are trained on the same training data set D

$$F(n) = \frac{1}{M}\Sigma_{i=1}^{M} F_i(n) \tag{9}$$

where $F_i(n)$ is the output of individual network i on the nth training pattern $\mathbf{x}(n)$, $F(n)$ is the output of the neural network ensemble on the nth training pattern, and M is the number of individual networks in the neural network ensemble.

The idea of negative correlation learning is to introduce a correlation penalty term into the error function of each individual network so that the mutual information among the ensemble can be minimized. The error function E_i for individual i on the training data set $D = \{(\mathbf{x}(1), y(1)), \cdots, (\mathbf{x}(N), y(N))\}$ in negative correlation learning is defined by

$$\begin{aligned} E_i &= \frac{1}{N}\Sigma_{n=1}^{N} E_i(n) \\ &= \frac{1}{N}\Sigma_{n=1}^{N}\left[\frac{1}{2}(F_i(n)-y(n))^2+\lambda p_i(n)\right] \quad (10) \end{aligned}$$

where N is the number of training patterns, $E_i(n)$ is the value of the error function of network i at presentation of the nth training pattern, and $y(n)$ is the desired output of the nth training pattern. The first term in the right side of Eq.(10) is the mean-squared error of individual network i. The second term p_i is a correlation penalty function. The purpose of minimizing p_i is to negatively correlate each individual's error with errors for the rest of the ensemble. The parameter λ is used to adjust the strength of the penalty.

The penalty function p_i has the form

$$p_i(n) = -\frac{1}{2}(F_i(n) - F(n))^2 \quad (11)$$

The partial derivative of E_i with respect to the output of individual i on the nth training pattern is

$$\begin{aligned} \frac{\partial E_i(n)}{\partial F_i(n)} &= F_i(n) - y(n) - \lambda(F_i(n) - F(n)) \\ &= (1-\lambda)(F_i(n) - y(n)) \\ &\quad +\lambda(F(n) - y(n)) \quad (12) \end{aligned}$$

where we have made use of the assumption that the output of ensemble $F(n)$ has constant value with respect to $F_i(n)$. The value of parameter λ lies inside the range $0 \leq \lambda \leq 1$ so that both $(1-\lambda)$ and λ have nonnegative values. The standard back-propagation (BP) [9] algorithm has been used for weight adjustments in the mode of pattern-by-pattern updating. That is, weight updating of all the individual networks is performed simultaneously using Eq.(12) after the presentation of each training pattern. One complete presentation of the entire training set during the learning process is called an *epoch*. Negative correlation learning from Eq.(12) is a simple extension to the standard BP algorithm. In fact, the only modification that is needed is to calculate an extra term of the form $\lambda(F_i(n) - F(n))$ for the ith neural network.

From Eqs.(10), (11), and (12), we may make the following observations:

1. During the training process, all the individual networks interact with each other through their penalty terms in the error functions. Each network F_i minimizes not only the difference between $F_i(n)$ and $y(n)$, but also the difference between $F(n)$ and $y(n)$. That is, negative correlation learning considers errors what all other neural networks have learned while training an neural network.

2. For $\lambda = 0.0$, there are no correlation penalty terms in the error functions of the individual networks, and the individual networks are just trained independently using BP. That is, independent training using BP for the individual networks is a special case of negative correlation learning.

3. For $\lambda = 1$, from Eq.(12) we get

$$\frac{\partial E_i(n)}{\partial F_i(n)} = F(n) - y(n) \quad (13)$$

Note that the error of the ensemble for the nth training pattern is defined by

$$E_{ensemble} = \frac{1}{2}(\frac{1}{M}\Sigma_{i=1}^{M} F_i(n) - y(n))^2 \quad (14)$$

The partial derivative of $E_{ensemble}$ with respect to F_i on the nth training pattern is

$$\begin{aligned} \frac{\partial E_{ensemble}}{\partial F_i(n)} &= \frac{1}{M}(\frac{1}{M}\Sigma_{i=1}^{M} F_i(n) - y(n)) \\ &= \frac{1}{M}(F(n) - y(n)) \quad (15) \end{aligned}$$

In this case, we get

$$\frac{\partial E_i(n)}{\partial F_i(n)} \propto \frac{\partial E_{ensemble}}{\partial F_i(n)} \quad (16)$$

The minimization of the error function of the ensemble is achieved by minimizing the error functions of the individual networks. From this point of view, negative correlation learning provides a novel way to decompose the learning task of the ensemble into a number of subtasks for different individual networks.

3 Simulation Results

In order to understand how negative correlation learning minimizes mutual information, this section analyses it through measuring mutual information on a regression task in three cases: noise free condition, small noise condition, and large noise condition.

3.1 Simulation Setup

The regression function investigated here is

$$f(\mathbf{x}) = \frac{1}{13}\left[10sin(\pi x_1 x_2) + 20\left(x_3 - \frac{1}{2}\right)^2 + 10x_4 + 5x_5\right] - 1 \quad (17)$$

where $\mathbf{x} = [x_1, \ldots, x_5]$ is an input vector whose components lie between zero and one. The value of $f(\mathbf{x})$ lies in the interval $[-1, 1]$. This regression task has been used by Jacobs [10] to estimate the bias of mixture-of-experts architectures and the variance and covariance of experts' weighted outputs.

Twenty-five training sets, $(\mathbf{x}^{(k)}(l), y^{(k)}(l))$, $l = 1, \cdots, L$, $L = 500$, $k = 1, \cdots, K$, $K = 25$, were created at random. Each set consisted of 500 input-output patterns in which the components of the input vectors were independently sampled from a uniform distribution over the interval (0,1). In the noise free condition, the target outputs were not corrupted by noise; in the small noise condition, the target outputs were created by adding noise sampled from a Gaussian distribution with a mean of zero and a variance of $\sigma^2 = 0.1$ to the function $f(\mathbf{x})$; in the large noise condition, the target outputs were created by adding noise sampled from a Gaussian distribution with a mean of zero and a variance of $\sigma^2 = 0.2$ to the function $f(\mathbf{x})$.

A testing set of 1024 input-output patterns, $(\mathbf{t}(n), d(n))$, $n = 1, \cdots, N$, $N = 1024$, was also generated. For this set, the components of the input vectors were independently sampled from a uniform distribution over the interval (0,1), and the target outputs were not corrupted by noise in all three conditions.

Each individual network in the ensemble is a multilayer perceptron with one hidden layer. All the individual networks have five hidden nodes in an ensemble architecture. The hidden node function is defined by the logistic function

$$\varphi(y) = \frac{1}{1 + \exp(-y)} \quad (18)$$

The network output is a linear combination of the outputs of the hidden nodes.

For each estimation of mutual information among an ensemble, twenty-five simulations were conducted. In each simulation, the ensemble was trained on a different training set from the same initial weights distributed inside a small range so that different simulations of an ensemble yielded different performances solely due to the use of different training sets. Such simulation setup follows the suggestions from Jacobs [10].

3.2 Measurement of Mutual Information

The average outputs of the ensemble and the individual network i on the nth pattern in the testing set, $(\mathbf{t}(n), d(n))$, $n = 1, \cdots, N$, are denoted respectively by $\overline{F}(\mathbf{t}(n))$ and $\overline{F}_i(\mathbf{t}(n))$, which are given by

$$\overline{F}(\mathbf{t}(n)) = \frac{1}{K}\Sigma_{k=1}^{K} F^{(k)}(\mathbf{t}(n)) \quad (19)$$

and

$$\overline{F}_i(\mathbf{t}(n)) = \frac{1}{K}\Sigma_{k=1}^{K} F_i^{(k)}(\mathbf{t}(n)) \quad (20)$$

where $F^{(k)}(\mathbf{t}(n))$ and $F_i^{(k)}(\mathbf{t}(n))$ are the outputs of the ensemble and the individual network i on the nth pattern in the testing set from the kth simulation, respectively, and $K = 25$ is the number of simulations. The correlation coefficient between network i and network j is given by

$$\rho_{ij} = \frac{\Sigma_{n=1}^{N}\Sigma_{k=1}^{K}\left(F_i^{(k)}(\mathbf{t}(n)) - \overline{F}_i(\mathbf{t}(n))\right)}{\sqrt{\Sigma_{n=1}^{N}\Sigma_{k=1}^{K}\left(F_i^{(k)}(\mathbf{t}(n)) - \overline{F}_i(\mathbf{t}(n))\right)^2}} \times \frac{\left(F_j^{(k)}(\mathbf{t}(n)) - \overline{F}_j(\mathbf{t}(n))\right)}{\sqrt{\Sigma_{n=1}^{N}\Sigma_{k=1}^{K}\left(F_j^{(k)}(\mathbf{t}(n)) - \overline{F}_j(\mathbf{t}(n))\right)^2}} \quad (21)$$

From Eq.(6), the integrated mutual information among the ensembles can be defined by

$$E_{mi} = -\frac{1}{2}\Sigma_{i=1}^{M}\Sigma_{j=1, j\neq i}^{M} log(1 - \rho_{ij}^2) \quad (22)$$

We may also define the integrated mean-squared error (MSE) on the testing set as

$$E_{test\_mse} = \frac{1}{N}\Sigma_{n=1}^{N}\frac{1}{K}\Sigma_{k=1}^{K}\left(F^{(k)}(\mathbf{t}(n)) - d(n)\right)^2 \quad (23)$$

The integrated mean-squared error E_{train} on the training set is given by

$$E_{train\_mse} = \frac{1}{L}\Sigma_{l=1}^{L}\frac{1}{K}\Sigma_{k=1}^{K}\left(F^{(k)}(\mathbf{x}^{(k)}(l)) - y^{(k)}(l)\right)^2 \quad (24)$$

3.3 Results in the Noise Free Condition

The results of negative correlation learning in the noise free condition for the different values of λ at epoch 2000 are given in Table 1. The results suggest that

Table 1: The Results of Negative Correlation Learning in the Noise Free Condition for Different λ Values at Epoch 2000.

λ	E_{mi}	E_{test_mse}	E_{train_mse}
0	0.3706	0.0016	0.0013
0.25	0.1478	0.0013	0.0010
0.5	0.1038	0.0011	0.0008
0.75	0.1704	0.0007	0.0005
1	0.6308	0.0002	0.0001

Table 2: The Results of Negative Correlation Learning in the Small Noise Condition for Different λ Values at Epoch 2000.

λ	E_{mi}	E_{test_mse}	E_{train_mse}
0	6.5495	0.0137	0.0962
0.25	3.8761	0.0128	0.0940
0.5	1.4547	0.0124	0.0915
0.75	0.3877	0.0126	0.0873
1	0.2431	0.0290	0.0778

both E_{train_mse} and E_{test_mse} appeared to decrease with increasing value of λ. The mutual information E_{mi} among the ensemble decreased as the value of λ increased when $0 \leq \lambda \leq 0.5$. However, when λ increased further to 0.75 and 1, the mutual information E_{mi} had larger values. The reason of having larger mutual information at $\lambda = 0.75$ and $\lambda = 1$ is that some correlation coefficients had negative values and the mutual information depends on the absolute values of correlation coefficients.

In order to find out why E_{train_mse} decreased with increasing value of λ, the concept of capability of a trained ensemble is introduced. The capability of a trained ensemble is measured by its ability of producing correct input-output mapping on the training set used, specifically, by its integrated mean-squared error E_{train_mse} on the training set. The smaller E_{train_mse} is, the larger capability the trained ensemble has.

3.3.1 Results in the Noise Conditions

Table 2 and Table 3 compare the performance of negative correlation learning for different strength parameters in both small noise (variance $\sigma^2 = 0.1$) and large noise (variance $\sigma^2 = 0.2$) conditions. The results show that there were same trends for E_{mi}, E_{test_mse}, and E_{train_mse} in both noise free and noise conditions when $\lambda \leq 0.5$. That is, E_{mi}, E_{test_mse}, and E_{train_mse} appeared to decrease with increasing value of λ. However, E_{test_mse} appeared to decrease first and then increase with increasing value of λ.

Table 3: The Results of Negative Correlation Learning in the Large Noise Condition for Different λ Values at Epoch 2000.

λ	E_{mi}	E_{test_mse}	E_{train_mse}
0	6.7503	0.0249	0.1895
0.25	3.9652	0.0235	0.1863
0.5	1.6957	0.0228	0.1813
0.75	0.4341	0.0248	0.1721
1	0.2030	0.0633	0.1512

In order to find out why E_{test_mse} showed different trends in noise free and noise conditions when $\lambda = 0.75$ and $\lambda = 1$, the integrated mean-squared error E_{train_mse} on the training set was also shown in Tables 1, 2, and 3. When $\lambda = 0$, the neural network ensemble trained had relatively large E_{train_mse}. It indicated that the capability of the neural network ensemble trained was not big enough to produce correct input-output mapping (i.e., it was underfitting) for this regression task. When $\lambda = 1$, the neural network ensemble trained learned too many specific input-output relations (i.e., it was overfitting), and it might memorize the training data and therefore be less able to generalize between similar input-output patterns. Although the overfitting was not observed for the neural network ensemble used in noise free condition, too large capability of the neural network ensemble will lead to overfitting for both noise free and noise conditions because of the ill-posedness of any finite training set [11].

Choosing a proper value of λ is important, and also problem dependent. For the noise conditions used for this regression task and the ensemble architectured used, the performance of the ensemble was optimal for $\lambda = 0.5$ among the tested values of λ in the sense of minimizing the MSE on the testing set.

4 Evolving Neural Network Ensembles

In EENCL [2], an evolutionary algorithm based on evolutionary programming [12] has been used to search for a population of diverse individual neural networks that solve a problem together. Two major issues were addressed in EENCL, including exploitation of the interaction between individual neural design and combination, and automatic determination of the number of individual neural networks in an ensemble. The major steps of EENCL are given as follows [4]:

1. Generate an initial population of M neural networks, and set $k = 1$. The number of hidden nodes for each neural network, n_h, is specified by the user. The random initial weights are distributed uniformly inside a small range.

2. Train each neural network in the initial population on the training set for a certain number of epochs using negative correlation learning. The number of epochs, n_e, is specified by the user.

3. Randomly choose a group of n_b neural networks as parents to create n_b offspring neural networks by Gaussian mutation.

4. Add the n_b offspring neural networks to the population and train the offspring neural networks using negative correlation learning while the remaining neural networks' weights are frozen.

5. Calculate the fitness of $M + n_b$ neural networks in the population and prune the population to the M fittest neural networks.

6. Go to the next step if the maximum number of generations has been reached. Otherwise, $k = k + 1$ and go to Step 3.

7. Form species using the k-means algorithm.

8. Combining species to form the ensembles.

There are two levels of adaptation in EENCL: negative correlation learning at the individual level and evolutionary learning based on evolutionary Forming species by using the k-means algorithm in EENCL [2] is not considered in this paper.

Fitness sharing used in EENCL is based on the idea of covering the same training patterns by shared individuals. The procedure of calculating shared fitness is carried out pattern-by-pattern over the training set. If one training pattern is learned correctly by p individuals in the population, each of these p individuals receives fitness $1/p$, and the rest of the individuals in the population receive zero fitness. Otherwise, all the individuals in the population receive zero fitness. The fitness is summed over all training patterns.

Rather than using the fitness sharing based on the covering set, a fitness sharing based on the minimization of mutual information was introduced in EENCL [6]. In order to create a population of neural networks that are as uncorrelated as possible, the mutual information between each individual neural network and the rest of population should be minimized. The fitness f_i of individual network i in the population can therefore be evaluated by the mutual information:

$$f_i = \frac{1}{\sum_{j \neq i} I(F_i, F_j)} \tag{25}$$

Minimization of mutual information has the similar motivations as fitness sharing. Both of them try to generate individuals that are different from others, though overlaps are allowed.

5 Experimental Studies

This section investigates EENCL with minimization of mutual information on two benchmark problems: the Australian credit card assessment problem and the diabetes problem. Both data sets were obtained from the UCI machine learning benchmark repository. They are available by anonymous ftp at ics.uci.edu (128.195.1.1) in directory /pub/machine-learning-databases.

The Australian credit card assessment problem is to assess applications for credit cards based on a number of attributes. There are 690 patterns in total. The output has two classes. The 14 attributes include 6 numeric values and 8 discrete ones, the latter having from 2 to 14 possible values.

The diabetes data set is a two-class problem that has 500 examples of class 1 and 268 of class 2. There are 8 attributes for each example. The data set is rather difficult to classify. The so-called "class" value is really a binarized form of another attribute that is itself highly indicative of certain types of diabetes but does not have a one-to-one correspondence with the medical condition of being diabetic.

In order to tell the difference between EENCL and EENCL with minimization of mutual information. We name the later approach as EENCLMI. The experimental setup is the same as the previous experimental setup described in [13, 2]. The n-fold cross-validation technique [14] was used to divide the data randomly into n mutually exclusive data groups of equal size. In each train-and-test process, one data group is selected as the testing set, and the other $(n-1)$ groups become the training set. The estimated error rate is the average error rate from these n groups. In this way, the error rate is estimated efficiently and in an unbiased way. The parameter n was set to be 10 for the Australian credit card data set, and 12 for the diabetes data set, respectively.

All parameters used in EENCLMI except for the number of training epochs were set to be the same for both problems: the population size M (25), the number of

Table 4: Comparison of Accuracy Rates between EENCLMI and EENCL for the Australian Credit Card Data Set. The Results Are Averaged on 10-Fold Cross-Validation. *Mean* and *SD* Indicate the Mean Value and Standard Deviation, Respectively.

	Simple Averaging		Majority Voting		Winner-Takes-All	
Methods	Mean	SD	Mean	SD	Mean	SD
EENCLMI	0.864	0.038	0.870	0.040	0.868	0.039
EENCL	0.855	0.039	0.857	0.039	0.865	0.028

Table 5: Comparison of Accuracy Rates between EENCLMI and EENCL for the Diabetes Data Set. The Results Are Averaged on 12-Fold Cross-Validation. *Mean* and *SD* Indicate the Mean Value and Standard Deviation, Respectively.

	Simple Averaging		Majority Voting		Winner-Takes-All	
Methods	Mean	SD	Mean	SD	Mean	SD
EENCLMI	0.771	0.049	0.777	0.046	0.773	0.051
EENCL	0.766	0.039	0.764	0.042	0.779	0.045

generations (200), the reproduction block size n_b (2), the strength parameter λ (0.5), the minimum number of cluster sets (3), and the maximum number of cluster sets (25). The number of training epochs n_e was set to 3 for the Australian credit card data set, and 15 for the diabetes data set. The used neural networks in the population are multilayer perceptrons with one hidden layer and five hidden nodes. These parameters were selected after some preliminary experiments. They were not meant to be optimal.

5.1 Experimental Results

Tables 4–5 show the results of EENCLMI for the two data sets, where the ensembles were constructed by the whole population in the last generation. Three combination methods for determining the output of the ensemble have been investigated in EENCLMI. The first is simple averaging. The output of the ensemble is formed by a simple averaging of output of individual neural networks in the ensemble. The second is majority voting. The output of the greatest number of individual neural networks will be the output of the ensemble. If there is a tie, the output of the ensemble is rejected. The third is winner-takes-all. For each pattern of the testing set, the output of the ensemble is only decided by the individual neural network whose output has the highest activation. The *accuracy rate* refers to the percentage of correct classifications produced by EENCLMI. In comparison with the accuracy rates obtained by three combination methods, majority voting and winner-takes-all outperformed simple averaging on both problems. Simple averaging is more suitable to the regression type of tasks. Because both problems studied in this paper are classification tasks, majority voting and winner-takes-all are better choices.

Tables 4–5 compare the results produced EENCLMI and EENCL using three combination methods. Majority voting supports EENCLMI, while winner-takes-all favors EENCL. Since the only difference between EENCLMI and EENCL is the fitness sharing scheme used, the results suggest that combination methods and fitness sharing are closely related to each other. Further studies are needed to probe the relationship of these two.

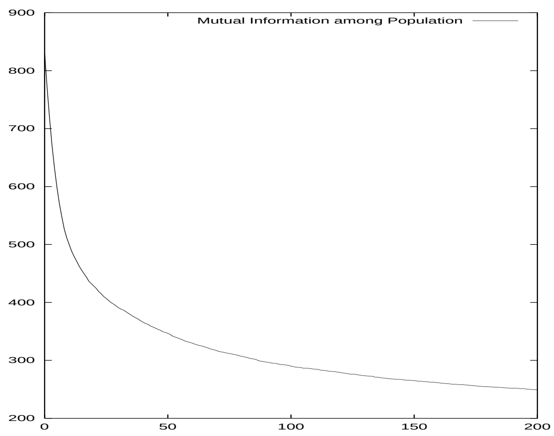

Figure 1: The Evolution of the Mean of Sum of the Mutual Information among the Population for the Australian Credit Card Data Set. The Mean Is Averaged on 10-fold Cross-Validation. The Vertical Axis Is the Mutual Information Value and the Horizontal Axis Is the Number of Generations.

In order to observe the evolutionary process of the mutual information among the population in EENCLMI, Figure 1 show the evolution of the mean of sum of the mutual information among the population for the Australian credit card data set. The sum of the mutual information among the population is calculated by

$$I_{population} = \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1, j \neq i}^{M} I(F_i, F_j) \qquad (26)$$

where F_i is the vector formed by the output of network i on the training set, and F_j is the vector formed by the output of network j on the training set. The mean of $I_{population}$ is averaged on 10-fold cross-validation. The evolutionary processes clearly shows that the value of mutual information among the population steadily decreased through the whole evolution.

6 Conclusions

Minimization of mutual information has been introduced as a fitness sharing scheme in EENCL. Compared with the fitness sharing based on the covering set originally used in EENCL [2], mutual information provides more accurate measurement on the similarity. By minimizing mutual information, a diverse population can be evolved.

This paper has also analyzed negative correlation learning in terms of mutual information on a regression task in the different noise conditions. Unlike independent training which creates larger mutual information among the ensemble, negative correlation learning can produce smaller mutual information among the ensemble.

References

[1] Y. Liu and X. Yao. Simultaneous training of negatively correlated neural networks in an ensemble. *IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics*, 29(6):716–725, 1999.

[2] Y. Liu, X. Yao, and T. Higuchi. Evolutionary ensembles with negative correlation learning. *IEEE Transactions on Evolutionary Computation*, 4(4):380–387, 2000.

[3] Y. Liu and X. Yao. Negatively correlated neural networks can produce best ensembles. *Australian Journal of Intelligent Information Processing Systems*, 4:176–185, 1998.

[4] X. Yao, Y. Liu, and P. Darwen. How to make best use of evolutionary learning. In R. Stocker, H. Jelinek, and B. Durnota, editors, *Complex Systems: From Local Interactions to Global Phenomena*, pages 229–242. IOS Press, Amsterdam, 1996.

[5] Y. Liu and X. Yao. Towards designing neural network ensembles by evolution. In *Parallel Problem Solving from Nature — PPSN V: Proc. of the Fifth International Conference on Parallel Problem Solving from Nature*, volume 1498 of *Lecture Notes in Computer Science*, pages 623–632. Springer-Verlag, Berlin, 1998.

[6] Y. Liu, Q. Zhao X. Yao, and T. Higuchi. Evolving a cooperative population of neural networks by minimizing mutual information. In *Proc. of the 2001 Conference on Evolutionary Computation*, pages 384–389. IEEE Press, 2001.

[7] J. C. A. van der Lubbe. *Information Theory*. Prentice-Hall International, Inc., 2nd edition, 1999.

[8] R. T. Clemen and R. .L Winkler. Limits for the precision and value of information from dependent sources. *Operations Research*, 33:427–442, 1985.

[9] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Vol. I*, pages 318–362. MIT Press, Cambridge, MA, 1986.

[10] R. A. Jacobs. Bias/variance analyses of mixture-of-experts architectures. *Neural Computation*, 9:369–383, 1997.

[11] J. H. Friedman. An overview of predictive learning and function approximation. In V. Cherkassky, J. H. Friedman, and H. Wechsler, editors, *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*, pages 1–61. Springer-Verlag, Heidelberg, Germany, 1994.

[12] D. B. Fogel. *Evolutionary Computation: Towards a New Philosophy of Machine Intelligence*. IEEE Press, New York, NY, 1995.

[13] D. Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood Limited, London, 1994.

[14] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, 36:111–147, 1974.