A New Learning Scheme for Neural Network Ensembles

Henrik Haraldsson and Mattias Ohlsson Complex Systems Division,
Department of Theoretical Physics
Lund University, Sölvegatan 14A,
SE-223 62 Lund, Sweden
Email: henrikh,mattias@thep.lu.se

Abstract

We propose a new method for training an ensemble of neural networks. A population of networks is created and maintained such that more probable networks replicate and less probable networks vanish. Each individual network is updated using random weight changes. This produces a diversity among the networks which is important for the ensemble prediction using the population. The method is compared against Bayesian learning for neural networks, Bagging and a simple neural network ensemble, on three datasets. The results show that the population method can be used as an efficient neural network learning algorithm.

I. Introduction

Ensemble learning for neural networks is a subject of active research. It enables an increase in generalization performance by combining several individual networks trained on the same task. The ensemble approach has been justified both theoretically [1], [2] and empirically [3]. The creation of an ensemble is often divided into two steps [4], the first being the judicious creation of the individual ensemble members and the second their appropriate combination to produce the ensemble output.

Combining the outputs is clearly only relevant when they disagree on some or several of the inputs. This insight was formalized by [2], who showed that the squared error of the ensemble when predicting a single target is equal to the average squared error of the individual networks, minus the diversity defined as the variance of the individual network outputs. Thus, to reduce the ensemble error, one tries to increase the diversity (called the “ambiguity” by Krogh and Vedelsby) without increasing the individual network errors too much.

The simplest method for creating diverse ensemble members is to train each network using randomly initialized weights. A more elaborate approach is to train the different networks on different subsets of the training set. Examples of this include Bagging [5] where each training set is created by resampling (with replacement) the original one, with uniform probability. Boosting [6] also uses resampling of the training set, but here the data points that were poorly classified by the previous networks receive a higher probability.

There also exist methods that explicitly try to maximize the disagreement among the ensemble members. The ADDEMUP algorithm [7], which uses genetic algorithms to create accurate members that disagree, was shown to perform well on four real-world applications. The idea of maximizing the diversity can also be found in the work of Liu [8]. Another approach was taken by Rosen [9], where an error correlation penalty term was added to the network error function to reduce the correlations of individual network errors. More developments and good selections of important ensemble research can be found in [4], [10] and [11].

Since neural network ensembles are easy to use and give better performance than single neural networks models, they are also used in real-world applications. From the medical domain one can find lung cancer cell identification [12], diagnosis of breast cancer [13], diagnosis of small round blue cell tumors using gene expression profiling [14] and detection of acute myocardial infarction using electrocardiograms [15], [16].

Here we present an approach that shares some of its philosophy from evolutionary neural networks [8] and genetic algorithms [17]. The key idea is to obtain set of disagreeing network members by initially populating different parts of state space. The method creates a *population* that consists of many neural networks. The network population is trained using Monte Carlo techniques with a Boltzmann distribution type of statistics. Each network evolves in state space by random weight updates together with a replication step where the most probable networks replicate into several identical copies. The Boltzmann distribution is used when finding probable networks. The ensemble output is obtained by simple averaging of the individual members. Some of the key features of the population method are:

- General approach, where almost any type of model and energy function for the population can be used.
- The training of the population members and the development of the population itself are intimately connected.

The population method was tested against traditional ensemble methods and Bayesian learning for neural network using hybrid Monte Carlo sampling [18], [19]. The test suite consisted of two real-world classification problems and one function approximation task. The results showed that our approach can produce neural network ensembles with good predictive performance, comparable with the other methods.

This paper is organized as follows: In section 2 we outline the *population* method and in section 3 some of its properties are discussed and exemplified numerically. Section 4 contains numerical explorations and comparisons. A summary and discussion is given in section 5.

II. Method

The purpose is to create a diverse ensemble of networks with good predictive performance. For this purpose, we consider a population of neural networks (in the order of 100 networks). During each iteration in the training process, networks evolve in state space by random moves, followed by a replication step where poorly adapted networks vanish and well-adapted networks replicate. Initially, poorly adapted networks have a relatively large probability to ”survive”. As the training continues, however, the selections gets stricter as well-adapted networks are favored more and more. The aim is to end up with a population that populates regions of state space where good networks are found.

First we consider the properties of individual networks; then we describe the population algorithm itself.

A. The ANN models

We consider neural networks in the form of feed-forward multilayer perceptrons (MLP) with one hidden layer and no direct input-output connections. Let $\vec{\omega}_i$ be the weight vector for each neural network (including thresholds). The hidden unit activation function is tanh(); the output activation function is linear for regression problems, logistic for binary classification, and softmax for classification with many classes. For regression, the targets are normalized during training. The inputs are always normalized.

For each model i we define an error function $E_{err}^{(i)}$ that measures the mismatch between the model and the training data. Let P denote the number of data points and K the number of outputs in the dataset. Possible choices of $E_{err}^{(i)}$ are:

$$E_{err}^{(i)} = \frac{1}{PK}\sum_{p=1}^{P}\sum_{k=1}^{K}\left(y_k^{(i)}(p) - t_k(p)\right)^2 \tag{1}$$

$$E_{err}^{(i)} = -\frac{1}{PK}\sum_{p=1}^{P}\sum_{k=1}^{K} t_k(p)\log y_k^{(i)}(p) \tag{2}$$

$$E_{err}^{(i)} = -\frac{1}{P}\sum_{p=1}^{P} t(p)\log y(p) + (1-t(p))\log(1-y(p)) \tag{3}$$

where $y_k^{(i)}(p)$ is the k:th output of network i using data point p as input, and $t_k(p)$ is the corresponding target. These error functions are used in regression, multi-value classification and binary classification, respectively.

In addition we introduce a weight elimination term [20], controlled by a tunable parameter λ, to regularize the network.

$$E_{reg}^{(i)} = \lambda \sum_c \frac{\omega_{ic}^2}{1+\omega_{ic}^2} \tag{4}$$

The sum for network i runs over the weights of the connections c between the layers, but not over the thresholds.

We define the total error, henceforth called the energy, of model i as $E_i = E_{err}^{(i)} + E_{reg}^{(i)}$.

B. Outline of the population method

The population method is executed by repeatedly performing steps 1-3 below.

0) Start with N ANN models that all are initialized by random weight vectors $\vec{\omega}_i$ $(i = 1, \ldots, N)$. As a time-saving preparation for the actual population method updates, the models are moved closer to the interesting regions by a rough back-propagation training for ∼5 epochs.
1) Update each $\vec{\omega}_i$ by making a random move: $\vec{\omega}_i \rightarrow \vec{\omega}_i + \epsilon\vec{\alpha}_i$, where $\vec{\alpha}_i$ is a vector of length 1 with a random direction and ϵ is a small number compared to unity.
2) Introduce a fictitious temperature T which determines the amount of competition between different models. Compute the energies E_i, Boltzmann factors B_i, average Boltzmann factor $\langle B\rangle$ and population (fitness) factors r_i:

$$B_i = e^{-E_i/T} \tag{5}$$

$$\langle B\rangle = \frac{1}{N}\sum_{i=1}^{N} B_i \tag{6}$$

$$r_i = \frac{B_i}{\langle B\rangle} = N\frac{e^{-E_i/T}}{\sum_j e^{-E_j/T}} \tag{7}$$

From equation 7 it follows that $\sum_i r_i = N$.

3) Write r_i as

$$r_i = L_i + \delta_i \tag{8}$$

where $L_i \geq 0$ is the integer part of r_i and δ_i is the remainder, $0 \leq \delta_i < 1$. Place L_i replicas of model i into the new population. Add one more replica with probability δ_i; this stochastic element among other things gives poorly fitted models (with $r_i < 1$) a chance of surviving. Do this for all models. In average this procedure will maintain N models.

The population size is constrained to be within $\pm$4% of the initial size. If the new population size is outside of these bounds, the proposed population is rejected and a new one is sampled. The $\pm$4% interval is wide enough to keep the rejection rate small, and still narrow enough that the size is roughly constant.

A completion of items 1-3 is called one iteration.

In the first part of the training, the temperature is annealed from the initial value T_i to the final value T_f by multiplying it each iteration by a constant factor < 1. The step size is also lowered from ϵ_i to ϵ_f in the same fashion.

After the annealing, the population continues to evolve according to items 1-3 before the actual network ensemble is sampled. This sampling is performed by recording all models in the population at S different times, with an interval of D_S iterations between each such “snapshot”. The default value of S is 10 to get a relatively dense sampling, while D_S is set so that the sampling is performed over the last two-thirds of the post-annealing phase. The final ensemble output $\langle y_k(p)\rangle$ is calculated simply by averaging over all the network outputs in the ensemble.

C. *Similar Approaches*

The proposed method is to some extent similar to genetic algorithms [17]. Genetic algorithms evolve a population of models, measure their performance by an objective function f_i and calculate the fitness as $f_i/\langle f\rangle$, whereupon a number of replicas are stochastically created in proportion to the fitness value. This approach is shared with the population method, for which $f_i = B_i = e^{-E_i/T}$. However, unlike genetic algorithms, the population method does not require its models to be represented as binary strings, and does not apply crossover operators to combine different models. In this respect, the population method is more similar to evolutionary programming.

Evolutionary programming is used by [8] to evolve a neural network ensemble of size N. In each iteration, a few extra networks are added by replicating some networks chosen with uniform probability, whereupon all new models make a rather large random move followed by back-propagation training. The training uses a cost term that explicitly favors diversity (cf. sections III-B.1 and V). Each iteration ends by pruning the the population to the N fittest networks. Though reminiscent of the population method, our method is different in the replication procedure and in the weight updates, which are random rather than gradient-based; the training is essentially performed by the replication step itself.

Another major difference lies in the annealed temperature parameter T that the fitness is based upon. This concept of annealing a temperature and considering a Boltzmann factor $e^{-E/T}$ is also used in simulated annealing [21], but in the latter method a new state is proposed and then accepted or rejected based on the difference in energy; a rejection means that the system returns to its previous state. In the population method, the new states are never rejected in the sense that the networks are returned to the state they had before the random move; rather the temperature is used in the selection between many such updated models.

III. Properties of the method

The properties and performance of the algorithm were evaluated using three test sets. The *Robot Arm* data set [22], [18] is an artificial regression problem, in which the outputs are simple trigonometric functions of the inputs, with a small Gaussian noise term added. The *Pima Indians Diabetes Database* [23] and *Myocardial Scintigram* data set [24] are medical classification problems with binary targets. The sizes of these data sets are specified in the Experiments section.

A. *Correlation over time*

It is important during all phases of the population learning that the ensemble of networks evolves in state space. Both in the beginning of the learning, in order to obtain well-tuned individual networks, and at the end where we want a diverse enough ensemble. To increase our confidence that the ensemble is evolving considerably, we find it suitable to measure that the population is uncorrelated with its state at a previous time.

To this end, we decided to study the average absolute activation (output) of the hidden units, $\langle |H|\rangle$. To our knowledge, this measure has not been studied by other authors. The average is formed both with respect to the hidden units and the data points in the training set. Averaging over the test set produces a very similar $\langle |H|\rangle$. Since the networks continually vanish and replicate, it seemed impractical to study the autocorrelation of individual networks; rather, we consider the average over all the networks in the population.

Our conjecture is that $\langle |H|\rangle$ will reflect the movement of our population in state space. A significantly fluctuating $\langle |H|\rangle$ indicates that the population is evolving rather than being stuck in a local minimum. However, the fluctuation of $\langle |H|\rangle$ cannot be used as a rule of thumb to indicate that the population has reached the desired distribution; even when we are training with less than optimal parameters, experiencing over-training etc, $\langle|H|\rangle$ still is usually fluctuating.

The qualitative behavior of $\langle|H|\rangle$ depends on the data set studied, and to some extent on the parameters chosen for training. $\langle|H|\rangle$ generally starts at ≈ 0.5 and then rises initially. When there is regularization ($\lambda \neq 0$), as for the Pima (Figure 1b) and Scintigram (Figure 1c) data sets, $\langle|H|\rangle$ then decreases considerably.

When the annealing is finished, the algorithm is continued, to allow for a detailed exploration. At this stage, with T and ϵ being low, the population cannot be expected to cross high energy barriers or travel long distances in state space. However, when training with the Pima and Scintigram data sets, the population is still mobile enough that $\langle|H|\rangle$ fluctuates significantly – Figure 1b and 1c. Interestingly, this is true even though the energy for the Pima data set is no longer changing in this phase (cf. Figure 4b).

Only for the Robot Arm, where the final values of T and ϵ are exceptionally low, are the fluctuations of $\langle|H|\rangle$ very small in the post-annealing phase (Figure 1a). These extreme parameters are required to get a fine-tuned answer for this low-noise problem.

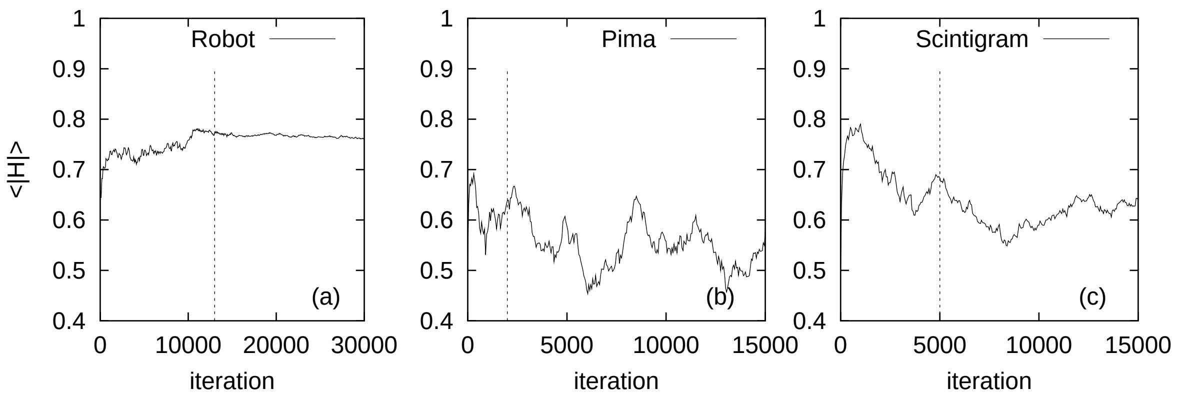

Fig. 1. The average absolute activation of the hidden units. (a) Robot Arm, (b) Pima and (c) Scintigram data sets. The vertical dashed line represents the iteration where the annealing was finished. (In order to show the long-term behavior of the hidden activation, these plots show a larger number of iterations than what was actually used when measuring the performance.)

As a closing remark regarding $\langle|H|\rangle$, it was noted that our chosen method of starting from a relatively large ϵ and then lowering ϵ gradually makes $\langle|H|\rangle$ rise more quickly to its final region, compared to using a small value of ϵ throughout. (A large constant ϵ cannot be used, as the final fine-tuning requires a small value.) This indicates that a large step size in the beginning of the training indeed makes the population explore larger regions in state space and reach the desired regions more quickly, as was the intention.

1) Autocorrelation as a stopping criterion: The question arises of how to determine when (and if) the ensemble has converged to the desired distribution. A stopping criterion, expressed in terms of autocorrelations of relevant model quantities, would certainly be desirable. An example of such a criterion would be to sample for 100 times the integrated correlation length τ, defined as

$$\tau = 1 + 2\sum_{s=1}^{\infty} \rho(s) \tag{9}$$

Here, $\rho(s)$ is the autocorrelation of the studied quantity $\xi(t)$ at lag s:

$$\rho(s) = \frac{E\left[(\xi(t) - E[\xi])\,(\xi(t-s) - E[\xi])\right]}{Var[\xi]}, \tag{10}$$

where t is the iteration number.

We made comparative runs with the Markov Chain Monte Carlo (MCMC) method implemented by Neal [18], [19], and drew the conclusion that for this algorithm as for our own, such a criterion is not reliable.

Neal's method uses Hamiltonian trajectories, discretized into a number of "leapfrog" steps, to sample from the Bayesian posterior distribution of network weights. These trajectories are alternated with Gibbs sampling of hyper-parameters controlling the prior weight distributions. In experiments using the Pima dataset, the smallest correlation lengths of hyper-parameters (measured in units of CPU minutes) were obtained for trajectories of 5 leapfrog steps. If the prescribed aim is to quickly sample for 100 times the correlation length, this would then be the ideal choice. But much longer trajectories are in fact needed to efficiently explore weight space; Neal uses trajectories in the order of 100–10000 steps. Thus we see that the proposed prescription is not reliable. Other authors [25] have performed other types of tests on Neal's method, and were also unable to establish convergence.

	Robot	Pima	Scintigram
Population method	0.00055	0.0012	0.0072
MCMC method	0.00049	0.0082	0.021
Bagging ensemble	—	0.0060	0.069
Bagging, unregularized	0.0011	0.13	0.093
Simple ensemble	0.0015	0.00071	0.00002

TABLE I
THE AVERAGE DIVERSITY $\bar{D}$ OF THE ENSEMBLES PRODUCED BY THE STUDIED METHODS.

For the Population method, the correlation length of $\langle|H|\rangle$ has the same order of magnitude as the total number of iterations found to be appropriate by cross-validation. This being so, it seems impossible in this case also to make a prescription to sample for a certain number of correlation lengths.

In conclusion, given the absence of such a measure of convergence for both these methods, the required number of iterations has to be determined by trial and error and self-consistency checks using cross-validation.

B. Correlation between networks at a given time, and in the final committee

1) Diversity: It is important that the ensemble members disagree in their predictions, if anything is to be gained by combining them. A measure of the disagreement is the ensemble diversity $\bar{d}(p)$ on input p, defined as

$$\bar{d}(p) = \frac{1}{N}\sum_{i=1}^{N}\Big(y^{(i)}(p) - \langle y(p)\rangle\Big)^2 \tag{11}$$

The diversity is just the variance of the individual network outputs. (For the moment we're considering the case of one output to keep notation simple, but the extension to many outputs is trivial.) Let $\bar{\epsilon}(p)$ denote the average of the individual network errors on input p, and $e_{err}(p)$ the squared error of the ensemble:

$$\bar{\epsilon}(p) = \frac{1}{N}\sum_{i=1}^{N}\Big(y^{(i)}(p) - t(p)\Big)^2 \tag{12}$$

$$e_{err}(p) = \left(\langle y(p)\rangle - t(p)\right)^2 \tag{13}$$

Then, as demonstrated in [2],

$$e_{err}(p) = \bar{\epsilon}(p) - \bar{d}(p) \tag{14}$$

Letting E_{err}, $\bar{\mathcal{E}}$ and $\bar{D}$ be the averages of e_{err}, $\bar{\epsilon}$ and $\bar{d}$ over the input distribution, we obtain

$$E_{err} = \bar{\mathcal{E}} - \bar{D} \tag{15}$$

In words, the mean square ensemble error is equal to the average squared error of the individual networks minus the average diversity. Thus, to get a small ensemble error we want the diversity to be large and the individual errors to be small. There is a trade-off; modifying an algorithm to increase the diversity will reduce the ensemble error only if the increase in diversity is larger than the increase in the individual errors. The average diversity $\bar{D}$ is an interesting measure of where the operation of an algorithm is on this scale, and the diversities of the population method along with the diversities of the other studied methods are given in Table I.

Some algorithms use an explicit mechanism for increasing the diversity; e.g. Bagging trains each network on a different subset of the training set. The population method does not use any explicit method for increasing diversity, and produces ensembles that are less diverse than Bagging, as is evident from Table I. However, there is not a one-to-one correspondence between diversity and good performance; even though the Bagging method produces more diverse ensembles than the population method for all three datasets, the predictive performance of the population method is higher (cf. section IV).

2) Internal correlation: Consider the vector of the activation of the hidden units, for a certain network and data point. We define the hidden activation distance between two networks to be the average Euclidian distance between the hidden activation vectors of these two networks, with the average being formed over the training set. This measure gives an indication of how different two network models are internally; cf. [26].

The hidden activation distance for the final population ensemble, for our choice of architecture and data sets, is typically in the range [0,0.2], whereas for independently trained and for randomly initialized networks it is typically in the range [0.5,3]. Thus, we see that all networks end up in the same approximate region of state space. One possible explanation is that, at some point in the annealing procedure, the group of networks in one region have happened to become significantly better adapted than in other regions, and the other groups vanish. Furthermore, even if the networks in each region are equally well adapted, the number of networks in any given group performs a random walk because of the randomness in the replication, and once a group reaches zero members it is gone.

What does this mean? Remember that there are numerous symmetries in neural networks – under permutation of the hidden unit indexes, and under sign change of all weights going into and out of one hidden unit. Because of this, there are many mutually distant regions in weight space that are symmetric copies of each other and implement essentially the same solution. Nothing is gained by including networks from many different symmetric regions; it is enough to explore one such region sufficiently. Thus, the fact that all models come from the same approximate region is not an essential disadvantage; they are still different enough that the outputs of the different models are significantly diverse (section III-B.1), which is important for ensemble performance.

The internal distance within the simple and Bagging ensembles is of course large, because these individually trained and randomly initialized networks have equal probability of ending up in either of the symmetric regions. Even though the internal distance is large, however, the diversity for the simple ensemble is small (Table I) and the predictive performance of the simple and Bagging ensembles is worse than for the Population method (section IV).

The internal distance between the networks in the MCMC ensemble is also large, because the Hamiltonian trajectories of Hybrid Monte Carlo travel long distances in state space. However, because many of the far-away regions are just symmetric copies of each other, visiting many of them is not really essential for good prediction. Doing so might still be an advantage, as an algorithm that only explores one symmetry-region runs the risk of exploring too small a part of that region, and moving between regions could protect against this problem. This is however at the cost of CPU-intensive calculations of the Hamiltonian equations of motion.

Neal rejects random updates in favor of hybrid Monte Carlo, arguing that random walk is an unsystematic and inefficient way of exploring state space [18]. This is indeed true if but a single network is used, but in the population algorithm the vanishing and replication of networks in effect moves poorly adapted networks long distances to the well-adapted ones where they continue their search, thus making the exploration systematic.

C. Age and Replication histograms

Each model has an age, defined as follows: in the beginning of the algorithm, all models have age 0. When a number of replicas is created from a model of age A, the first of these replicas has age $A + 1$ and the rest (if any) have age 1. Figure 2 shows a histogram of how old the models were when they vanished (i.e. were replicated zero times), during a window of iterations at different stages of the algorithm. Figure 3 correspondingly shows a histogram of how many replicas were created from each model.

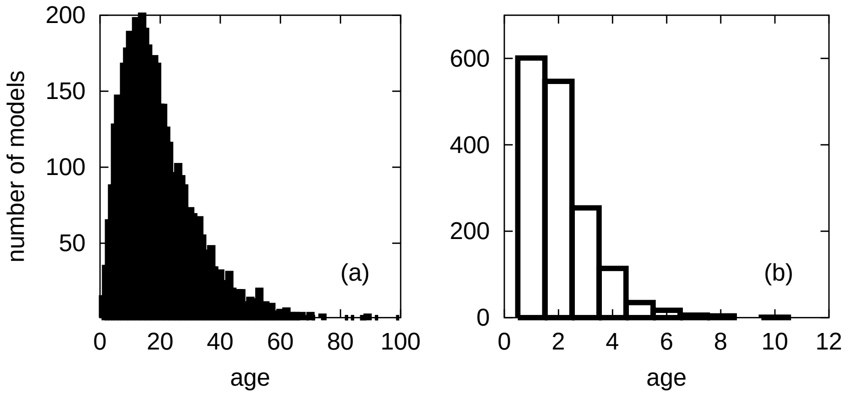

Fig. 2. The age distribution of the models when they vanish, for the Robot Arm data set. (a) First 300 iterations. The competition is low and some models grow very old. (b) Last 10 iterations; the competition is strong.

At the beginning of the procedure, T is high. Our way of normalizing the targets (for regression) and the error means that the approximate magnitude of the initial energy is the same for all data sets, and the default value $T_i = 3.0$ is usually appropriate. A high T implies that all $r_i \approx 1$, and thus most models are replicated exactly once every iteration, as can be seen in Figure 3a. This in turn means that there are many models that reach a high age. The typical lifetime of a model is 10–30 as its population factor fluctuates around unity, and some models survive for as much as 100 iterations.

As T is lowered, the tail of the replication histogram grows longer and longer as the best models are more and more favored. When the annealing is almost finished, T is very low which means that the competition is strong. The energies of the models lie within a rather narrow band; a rather small change in energy is enough to give a model a tiny or huge population factor. The energy is still decreasing, i.e. often there is a model that finds a better configuration than what's been achieved before. This model is then replicated a very large number of times – Figure 3b.

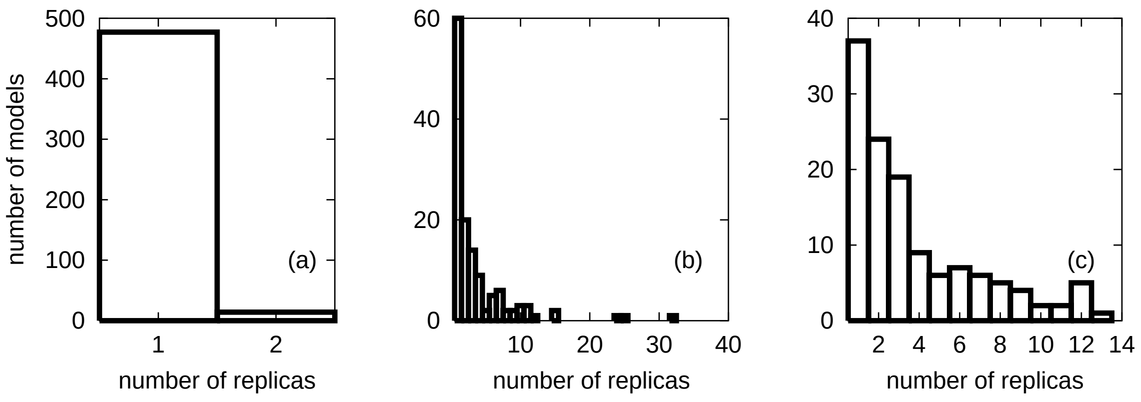

Fig. 3. Replication histograms, showing the number of replicas produced by the replicating networks, for the Robot Arm data set. (a) Start of training. (b) End of annealing; the energy is still decreasing. (c) End of training.

	Robot	Pima	Scintigram
Number of hidden units	10	8	8
Population size	500	100	100
Number of annealing iterations	13000	2000	5000
Total number of iterations	16000	2600	5500
Weight elimination parameter	0	5.0	20.0
Initial stepsize	0.1	0.5	0.5
Final stepsize	0.035	0.07	0.06
Initial temperature	0.08	3.0	3.0
Final temperature	0.00006	0.001	0.001

TABLE II
TRAINING PARAMETERS USED FOR THE POPULATION METHOD.

After the annealing is finished, the energy eventually stops decreasing, and the population spreads out over the best regions found. The competition is still strong, but as there is now a fair number of networks that have reached very low and similar energies, no network is favored greatly over the others and the maximum number of replicas is moderate – Figure 3c. Remember that each individual model performs a random walk, and it might easily move to a configuration with significantly higher energy, after which the model is likely to vanish. Thus, we cannot expect models to become all that old. This can be seen in Figure 2b, which shows the ages during the last 10 iterations.

The final temperature T_f needed to archive good performance is problem dependent; there is a trade-off between forcing all the models to have very low energies by setting a low T_f, and increasing the diversity by setting T_f higher. As a rule of thumb we used $T_f = 10^{-3}$.

IV. EXPERIMENTS

To evaluate the performance of the Population algorithm we compared against the MCMC method by Neal [18], [19], Bagging [5] and a simple averaging ensemble, formed by training 100 MLPs on the full training set using back-propagation with random weight initialization. We also used 100 networks for the Bagging ensemble.

In preliminary runs, parameters were chosen using trial and error based on three-fold cross-validation on the training set. Reasonable effort was used in finding suitable parameters to minimize the error and CPU consumption, but the purpose here is only to give an indication of the merits of the methods, not to make a rigorous test of the general performance. The chosen parameters for the Population method are specified in Table II. The parameters used for the other methods are not specified here, but the number of hidden units used for a given data set was the same for all four methods. Other values for the number of hidden units were tested for the four methods, but this did not increase predictive performance. Furthermore, much larger ensemble sizes were tried for all methods, without any increase in performance. Apart from the Robot Arm data set that requires unusual parameters, the ones specified in Table II will also serve as an appropriate set of initial parameters for other problems. However, parameters such as the one for the weight elimination always requires tuning for each specific data set.

For the classification tasks, the performance is given in terms of area under the receiver operating characteristic (ROC) curve. The ROC curve displays diagnostic accuracy expressed in terms of sensitivity against 1 - specificity at all possible output threshold values. The sensitivity is the fraction of correctly classified ”one” cases and the specificity is the corresponding fraction for the ”zero” cases. The area under the ROC curve is a commonly used performance measure with unity (100%) being a perfect classifier and the value 0.5 being equivalent to random guessing. The total performance, i.e. the fraction of

	Robot	**Pima**		**Scintigram**	
Problem type	Regression	Classification		Classification	
Number of inputs	2	7		30	
Number of targets	2	1		1	
Size of training set	200	200		153	
Size of test set	200	332		76	
	Mean Square Error	ROC area (%)	Total perf. (%)	ROC area (%)	Total perf. (%)
Population method	0.00308(11)	86.51(4)	80.2(2)	80.8(1.0)	75.5(1.3)
MCMC method	0.00279(2)	86.30(13)	80.1(3)	80.8(7)	75.1(1.2)
Bagging ensemble	—	86.18(7)	79.3(2)	78.4(7)	71.7(1.3)
Bagging, unregularized	0.00335(4)	83.52(16)	77.8(6)	77.2(4)	71.3(1.4)
Simple ensemble	0.00367(7)	86.11(4)	79.6(4)	75.5(1)	75.0(0)
	CPU minutes	CPU minutes		CPU minutes	
Population method	435	19		93	
MCMC method	58	79		386	
Bagging ensemble	3.3	2.9		4.6	
Simple ensemble	3.8	2.8		3.2	

TABLE III

PROPERTIES OF THE DATA SETS (TOP); TEST RESULTS (MIDDLE), AND CPU CONSUMPTION (BOTTOM). THE STANDARD DEVIATIONS, CORRESPONDING TO THE ONE OR TWO LAST DIGITS OF THE TEST RESULTS, ARE GIVEN WITHIN PARENTHESES.

correctly classified examples in the test set, is also given at the threshold level that maximized the fraction of correctly classified examples during cross-validation. For the regression task, performance is given in terms of the mean square error (equation 1).

A. *Test results*

A final training was made using the whole training set, and the performance was computed using the test set. In each case, ten runs with identical parameters were started. The results were stable; the performances with standard deviations are as specified in Table III. All runs were made on a 800 MHz Pentium III computer running Linux.

The Robot Arm is an artificial data set with very low noise. When starting the population updates from random configurations with high energies, the population often did not reach the lowest energies during the annealing. Therefore a very long initial back-propagation training was used, in order to start with a low energy. The initial temperature was also low; otherwise there would have been practically no competition between the networks as they are randomly updated, and the energy would have increased rapidly. The fine-tuning required of the networks for this problem necessitated an extremely low final temperature, which means that the population is relatively fixed after the annealing is finished, with the hidden activation $\langle|H|\rangle$ almost constant (Figure 1a). The energy also settles and fluctuates within a rather narrow interval; Figure 4a. — Even though these unusual parameters were used, the MCMC method achieves a better result in this case.

The Pima dataset is a real-world problem requiring less extreme parameters. As can be seen in Figure 4b, the training and test errors settle after the annealing and stay practically constant. This does not mean that the population is stuck in a local minimum, however; unlike for the Robot Arm, $\langle|H|\rangle$ is fluctuating (Figure 1b), indicating that the population is changing significantly. The performance for the Population and MCMC method is practically the same.

When cross-validating with the Scintigram dataset, there was often over-training; the validation error decreased initially but often subsequently increased; cf. Figure 4c. The validation curve could be made to level out by increasing the weight elimination parameter λ, but this resulted in worse predictive performance. This problem was solved by early stopping. An overall validation curve was created by averaging over the validation curves of several cross-validation runs with different dataset partitions; the best region of this curve was found to be the interval 1900–5500. In the final run, this was the interval over which the ensemble was sampled. Again, the performance of the Population and MCMC method is the same.

The tested ensemble methods used a non-zero regularization term for the two classification problems, as a result of the cross-validation process. As a reference we have included large unregularized ensembles (500 networks) created with Bagging. As can be seen in Table III, the performance for the unregularized Bagging ensembles on the classification tasks were worse than for the regularized ones.

In all, the results for the Population method and Neal's Bayesian software were comparable for the real-world classification tasks, whereas Neal's software was better than the Population method on the artificial Robot Arm data set. The Population method was however faster than the MCMC method on the classification problems. Compared to the two other ensemble methods, Bagging and the simple ANN ensemble, the Population method was better, at least for the Robot and the Scintigram datasets. Bagging and the simple ANN ensemble were on the other hand the overall fastest methods.

V. SUMMARY AND DISCUSSION

We have presented a new method for training an ensemble of neural networks. A population of networks is created and maintained such that probable networks replicates and less probable networks vanish. The Boltzmann distribution is used

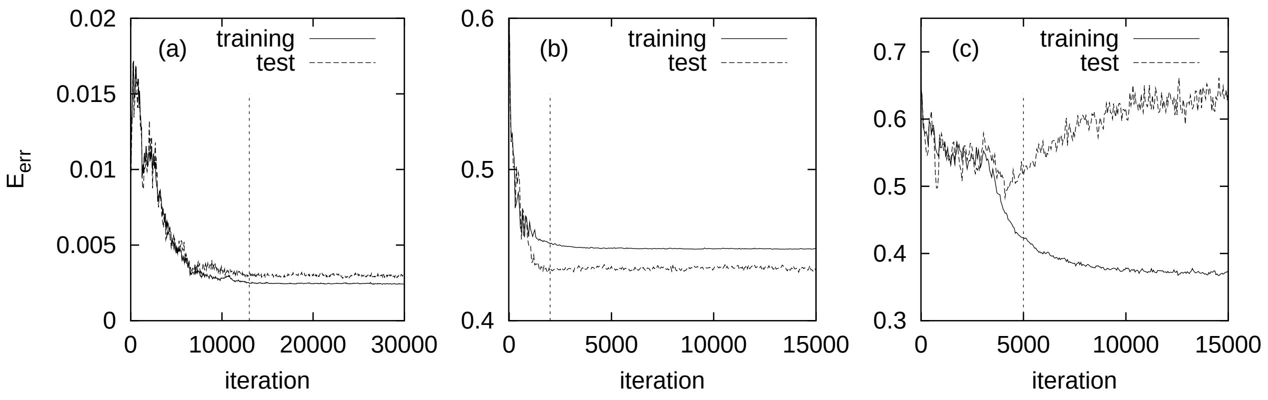

Fig. 4. Training curves: (a) Robot Arm. (b) Pima. (c) Scintigram. These training curves show more iterations than were actually used when measuring performance, as in Figure 1.

when defining the probability of a network. The “temperature” appearing in the Boltzmann distribution signifies the level of competition among the networks, where a high temperature allows for networks that have both small and large errors. A small temperature, on the other hand, selects only well adapted networks. The annealing present in the population method is used to transform an initial (random) population to a set of fine-tuned networks. This population then continues to evolve in state space. The whole procedure creates the necessary diversity among the individual members that is important for the ensemble itself.

The method is simple in the sense that is does not require any complicated algorithmic implementations. Furthermore, no gradient information, with respect to the error function, is used when updating individual networks, which makes the method fast even when maintaining a population of a hundred networks or more.

We tried using an explicit mechanism for increasing the diversity of the population, by introducing a term in the energy equal to the diversity $\bar{D}$ (equation 11) times a negative, tunable parameter. Such a term has been used by other authors when training new networks that are to be added to an existing ensemble [8], or when selecting the best networks trained by other means [7]. For our method, this term did not improve the predictive performance for any of the studied datasets. The only noticeable non-detrimental effect was on the Pima dataset, where moderate values of the tunable parameter resulted in doubled average diversity $\bar{D}$ at unchanged ROC area. In conclusion, nothing is gained in the population algorithm by trying to push the trade-off between diversity and individual network errors in this way.

To summarize, the algorithm has been tested on three datasets and the results from the experiments section indeed show that the population method can be used as an efficient neural network learning algorithm.

VI. Acknowledgments

We thank Carsten Peterson for fruitful discussions. This work was in part supported by the Swedish Natural Science Research Council and the Swedish Foundation for Strategic Research.

References

[1] L. K. Hansen and P. Salamon, “Neural network ensembles,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, pp. 993–1001, 1990.

[2] A. Krogh and J. Vedelsby, “Neural network ensembles, cross validation, and active learning,” in *Advances in Neural Information Processing Systems*, G. Tesauro, D. Touretzky, and T. Leen, Eds., vol. 2. San Mateo, CA: Morgan Kaufman, 1995, pp. 650–659.

[3] D. Opitz and R. Maclin, “Popular ensemble methods: An empirical study,” *Journal of Artificial Intelligence Research*, vol. 11, pp. 169–198, 1999.

[4] A. J. C. Sharkey, “On combining artificial neural nets,” *Connection Science*, vol. 8, pp. 299–314, 1996.

[5] L. Breiman, “Bagging predictors,” *Machine Learning*, vol. 24, pp. 123–140, 1996.

[6] Y. Freund and R. E. Schapire, “Experiments with a new boosting algorithm,” in *Machine Learning: Proceedings of the Thirteenth International Conference*, 1996, pp. 148–156.

[7] D. W. Opitz and J. W. Shavlik, “Actively searching for an effective neural-network ensemble,” *Connection Science*, vol. 8, pp. 337–353, 1996.

[8] Y. Liu, X. Yao, and T. Higuchi, “Evolutionary ensembles with negative correlation learning,” *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 3, pp. 295–304, 2000.

[9] B. E. Rosen, “Ensemble learning using decorrelated neural networks,” *Connection Science*, vol. 8, pp. 373–384, 1996.

[10] A. Sharkey, Ed., *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems*. London: Springer-Verlag, 1999.

[11] V. Tresp, “Committee machines,” in *Handbook for Neural Network Signal Processing*, Y. H. Hu and J.-N. Hwang, Eds. CRC Press, 2001.

[12] Z.-H. Zhou, Y. Jiang, Y.-B. Yang, and S.-F. Chen, “Lung cancer cell identification based on artificial neural network ensembles,” *Artificial Intelligence in Medicine*, vol. 24, no. 1, pp. 25–36, 2002.

[13] A. J. C. Sharkey, N. E. Sharkey, and S. S. Cross, “Adapting an ensemble approach for the diagnosis of breast cancer,” in *Proceedings of ICANN 98*. Springer-Verlag, 1998, pp. 281–286.

[14] J. Khan, J. Wei, M. Ringnér, L. Saal, M. Ladanyi, F. Westermann, F. Berthold, M. Schwab, C. Atonescu, C. Peterson, and P. Meltzer, “Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks,” *Nature Medicine*, vol. 7, pp. 673–679, 2001.

[15] M. Ohlsson, H. Öhlin, S. Wallerstedt, and L. Edenbrandt, “Usefulness of serial electrocardiograms for diagnosis of acute myocardial infarction,” *The American Journal of Cardiology*, vol. 88, pp. 478–481, 2001.

[16] S.-E. Olsson, H. Öhlin, M. Ohlsson, and L. Edenbrandt, “Neural networks - a diagnostic tool in acute myocardial infarction with concomitant left bundle branch block,” *Clinical Physiology and Functional Imaging*, vol. 22, pp. 295–299, 2002.

[17] D. Whitley, “A genetic algorithm tutorial,” *Statistics and Computing*, vol. 4, pp. 65–85, 1994.

[18] R. M. Neal, *Bayesian Learning for Neural Networks*. New York: Springer-Verlag, 1996.

[19] ——, “Software for flexible Bayesian modeling,” 1999, http://www.cs.toronto.edu/˜radford/fbm.1999-03-13.doc/index.html.

[20] S. J. Hanson and L. Y. Pratt, “Comparing biases for minimal network construction with back–propagation,” in *Advances in Neural Information Processing Systems 1*, D. S. Touretzky, Ed. Morgan Kaufmann, 1989, pp. 177–185.

[21] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by simulated annealing,” *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[22] D. J. C. MacKay, “A practical Bayesian framework for backpropagation networks,” *Neural Computation*, vol. 4, no. 3, pp. 448–472, 1992.

[23] B. D. Ripley, *Pattern Recognition and Neural Networks*. Cambridge, Great Britain: Cambridge University Press, 1996.

[24] H. Haraldsson, M. Ohlsson, and L. Edenbrandt, “Value of exercise data for the interpretation of myocardial perfusion SPECT,” *Journal of Nuclear Cardiology*, vol. 9, pp. 169–173, 2002.

[25] F. Vivarelli and C. K. I. Williams, “Comparing Bayesian neural network algorithms for classifying segmented outdoor images,” *Neural Networks*, vol. 14, no. 4–5, pp. 427–437, 2001.

[26] R. Caruana, “Case-based explanation for artificial neural nets,” in *Artificial Neural Networks in Medicine and Biology*, H. Malmgren, M. Borga, and L. Niklasson, Eds. London: Springer-Verlag, 2000, pp. 303–308.