Evolution of an Artificial Neural Network Based Autonomous Land Vehicle Controller

Shumeet Baluja

Abstract— **This paper presents an evolutionary method for creating an artificial neural network based autonomous land vehicle controller. The evolved controllers perform better in unseen situations than those trained with an error backpropagation learning algorithm designed for this task. In this paper, an overview of the previous connectionist based approaches to this task is given, and the evolutionary algorithms used in this study are described in detail. Methods for reducing the high computational costs of training artificial neural networks with evolutionary algorithms are explored. Error metrics specific to the task of autonomous vehicle control are introduced; the evolutionary algorithms guided by these error metrics reveal improved performance over those guided by the standard sum-squared error metric. Finally, techniques for integrating evolutionary search and error backpropagation are presented. The evolved networks are designed to control Carnegie Mellon University's NAVLAB vehicles in road following tasks.**

I. Introduction

SPECIAL purpose hardware is currently being designed at Carnegie Mellon University to allow pre-trained artificial neural networks (ANN's) to control the steering direction of an autonomous vehicle. The hardware design does not support modification of the network weights; therefore, on-line retraining of the networks is not possible. However, current neural network based methods for controlling autonomous land vehicles require frequent retraining to adapt to changing road, weather, and lighting conditions. An alternative to retraining is to use a pool of pre-trained networks in which each network is trained to work well in a different situation. Because each of the networks in the pool will be trained only once (before the system is installed), the generalization ability of these networks is crucial to good performance. The goal of this project is to create networks which have good generalization capabilities; they should be able to perform well in scenes which may be similar, but not identical, to those used in training.

In this paper, an evolutionary method for developing artificial neural networks for autonomous vehicle control is explored. The standard method of training ANN's, error backpropagation, is a method of gradient descent through the weight space, and is therefore susceptible to getting caught in local minima. In contrast, evolutionary algorithms (EA's) are global search heuristics, and should be less susceptible to local minima.

There currently exists a great deal of interest in both evolutionary optimization techniques and artificial neural networks within the artificial intelligence communities. Recently, many attempts have been made to integrate the two fields. This integration has broadly moved along two axes, that of applying connectionist principles to evolutionary procedures in order to enhance the capabilities of the evolutionary procedures [1], [4], and applying evolutionary principles to enhance the capabilities of artificial neural networks (ANN's) [2], [5], [6], [9], [10], [14], [20], [22], [29], [31]. This paper concentrates on the latter form of integration; evolutionary optimization methods are used to improve the generalization capabilities of feed-forward artificial neural networks.

Many of the previous studies involving evolutionary optimization techniques applied to ANN's have concentrated on relatively small problems. This paper presents a study of an evolutionary optimization method on a real-world problem, that of autonomous navigation of Carnegie Mellon's NAVLAB system. In contrast to the other problems addressed by similar methods in recently published literature, this problem has a large number of pixel-based inputs, and on one of the two tasks presented in this paper, also has a large number of outputs to indicate the appropriate steering direction.

The feasibility of using evolutionary algorithms for network topology discovery and weight optimization is discussed throughout the paper. Methods for avoiding the high computational cost associated with these procedures are presented. Nonetheless, evolutionary algorithms remain more computationally expensive than training by standard error backpropagation. Because of this limitation, the ability to train on-line, which may be important in many real-time reactive robotic environments, is not addressed in this paper. In the current autonomous vehicle system, (re)training speed is crucial because the system must be able to adapt "on-the-fly" to changing conditions. In the system explored in this paper, on-line training is not required, as the full network pool is trained before any of the networks are used.

In this paper, error metrics specific to the task of autonomous vehicle control are introduced; the evolutionary algorithms guided by these error metrics reveal improved performance over those guided by the standard sum-squared error metric. Also explored in this paper are methods of integrating

Manuscript received June 10, 1994; revised December 31, 1994, February 22, 1995, and March 11, 1995. This work was supported by a National Science Foundation Graduate Fellowship, and was partially sponsored by the Department of Transportation/National Highway Traffic Safety Administration, under Contract DTNH22-93-C-07023 titled "IVHS Run-Off-Road Countermeasures". The views and conclusions contained in this document are those of the author and should not be interpreted as representing official policies, either expressed or implied, of the National Science Foundation, the Department of Transportation, or the U.S. Government.

The author is with the School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 15213 USA (e-mail: baluja@cs.cmu.edu; http://www.cs.cmu.edu/~baluja).

Publisher Item Identifier S 1083-4419(96)03242-6.

Fig. 1. The Carnegie Mellon NAVLAB autonomous navigation testbed.

evolutionary search and backpropagation. The integration of these two methods yields slightly better results than those achieved by evolutionary search alone.

The next section describes the ANN based autonomous land vehicle controller, ALVINN (Autonomous Land Vehicle In a Neural Network), which serves as the prototype for the studies presented here [25]. Section III describes the evolutionary approach used to evolve the neuro-controller. Section IV presents an overview of the experiments. Section V compares standard and evolutionary approaches to this problem. Section VI examines the use of a task-specific error metric for guiding the evolution. The paper concludes with a discussion of the results and future directions for research.

II. Introduction to ALVINN

ALVINN is an artificial neural network based perception system which learns to control Carnegie Mellon's NAVLAB vehicle, see Fig. 1, by watching a person drive. ALVINN's architecture consists of a single hidden layer backpropagation network. The input layer of the network is a 30 × 32 unit 2-D "retina" which receives input from the vehicle's video camera, see Fig. 2. Each input unit is fully connected to a layer of four hidden units which are in turn fully connected to a layer of 30 output units. In the simplest interpretation, each of the network's output units can be considered to represent the network's vote for a particular steering direction. After presenting an image to the input retina, and passing activation forward through the network, the activations of the output units represent the steering arc the network believes to be best for staying on the road.

To teach the network to steer, ALVINN is shown video images from the onboard camera as a person drives, and is trained to output the steering direction in which the person is currently steering. ALVINN is able to learn which image features are important for particular driving situations. It has been successfully trained to drive in a wider variety of situations than other autonomous navigation systems which require fixed, predefined features (e.g., the road's center line) for accurate driving. The situations ALVINN networks have been trained to handle include single lane dirt roads, single lane paved bicycle paths, two lane suburban neighborhood streets, and lined divided highways. In this last domain, ALVINN has

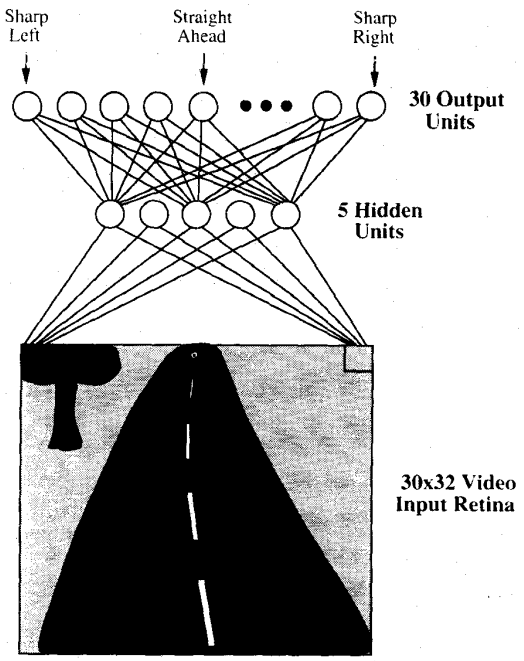


Fig. 2. The ALVINN neural network architecture.

successfully driven autonomously at speeds of up to 55 miles per hour, and for distances of over 90 miles on a highway north of Pittsburgh, PA [25], [27].

The performance of the ALVINN system has been extensively analyzed by Pomerleau [23], [24], [25]. Throughout the development of ALVINN, various network architectures have been examined, including architectures with more hidden units and different output representations. Although the output representation was found to have a large impact on the effectiveness of the network, other features of the network architecture were found to yield approximately equivalent results [25]. In this paper, two output representations will be considered; the first representation uses only a single output to determine the steering direction. The second representation is the one currently used in the ALVINN system, a distributed output representation of 30 units. There are three motivating factors in choosing these output representations for study. The first is that the hardware design uses the 30 output unit representation: the networks developed in this study should be compatible with this hardware. The second motivation is that the 30 unit representation has been successfully used in the ALVINN/NAVLAB system. The third reason is that previous studies of both these representations provide benchmarks with which to compare new results. The output representations are described in greater detail in Section IV.

A. Training ALVINN

To train ALVINN, the network is presented with road images as input and the corresponding correct steering direction as the desired output. The backpropagation algorithm alters the strengths of the connections between the units so that the network produces the appropriate steering response

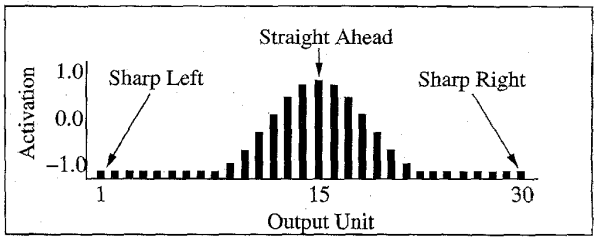


Fig. 3. Gaussian output representation, using 30 output units. The steering direction represented in the above diagram is straight ahead. Figure adapted from [25].

when presented with a video image of the road ahead of the vehicle. Training is currently done on-line with an onboard Sun SPARC-10 workstation.

Several modifications to the standard backpropagation algorithm are used to train ALVINN. First, the weight change momentum factor is steadily increased during training. Second, the learning rate constant for each weight is scaled by the fan-in of the unit to which the weight projects. Third, a large amount of neighbor weight smoothing is used between the input and hidden layers. Neighbor weight smoothing is a technique to constrain weights which connect to neighboring units in the input retina to similar values. This is a method of preserving spatial information in the context of the backpropagation algorithm [25].

In its current implementation, ALVINN is trained to produce a Gaussian distribution of activation centered around the appropriate steering direction, see Fig. 3. By requiring the network to produce a Gaussian distribution as output, instead of a more traditional "one out of N" classification, the learning task is made easier since slightly different road images require the network to respond with only slightly different output vectors. This is in contrast to the highly nonlinear output requirement of the "one out of N" representation in which the network must significantly alter its output vector (from having one unit on and the rest off, to having a different unit on and the rest off) on the basis of fine distinctions between slightly shifted road scenes [25].

One of the problems associated with training ALVINN is that the human driver will normally steer the vehicle correctly down the center of the road (or lane). Therefore, the network will never be presented with situations in which it must recover from errors, such as being off the correct portion of the road. In order to compensate for this lack of real training data, the images are shifted by various amounts relative to the road's center. The shifting mechanism maintains the correct perspective to ensure that the shifted images are realistic. The correct steering direction is determined by the amount of shift introduced into the images. A description of the shifting mechanism can be found in [23], [25]. The network is trained on the original and shifted images.

III. An Evolutionary Approach

Evolutionary algorithms (EA's) span a very broad range of learning algorithms which rely on a population of individuals, each of which represents a search point in the space of potential solutions to the given problem [11]. In one of the standard EA techniques, genetic algorithms (GA's) [17], the population of points, which is usually initialized randomly, evolves toward better regions of the search space by means of selection, recombination and mutation. The environment, which is defined by the specific problem to be optimized, returns information assessing the quality of each of the individual search points. The selection process favors the survival of the points with a higher assessed quality. The recombination operator allows several members of the current population of points to contribute to a member of the subsequent generation. The mutation operator ensures that the population does not become too homogenous. Homogeneity decreases the ability of the GA to perform innovative search. A comprehensive overview of evolutionary algorithms, and their applications to parameter optimization tasks, can be found in [3], [12].

The majority of approaches in which evolutionary principles are used in conjunction with neural network training can be broadly subdivided into two groups. The first concentrates on formulating the problem of finding the connection weights of a pre-defined artificial neural network architecture as a search problem. Traditionally backpropagation, or one of its many variants, has been used to train the weights of the connections. However, backpropagation is a method of gradient descent through the weight space, and can therefore get stuck in local minima. Evolutionary algorithms are global search heuristics, and are less susceptible to local minima. Finding the appropriate set of weights in a neural network can be formulated as a parameter optimization problem to which EA's can be applied in a straightforward manner.

The second group uses EA's to find the appropriate structure of the network for the particular task; the number of layers, the connectivity, etc., are defined through the search process. The weights can either be determined using a separate neural network training algorithm, or can simultaneously be found while searching for the network topology.

The approach used in this paper is a variant of the second group of approaches described above; it will be presented in greater detail in Section V. One of the advantages of this method is that if there is very little knowledge of the structure of the problem, or if no knowledge other than the number of inputs and outputs needed is available, the structure of the network does not need to be predefined in detail.

Given the possibility of backpropagation falling into a local minimum, and the potential lack of knowledge regarding the appropriate neural network architecture to use, using EA's appears to be an interesting option. Previous studies which have used evolution as the principal learning paradigm for training artificial neural networks have often modeled evolution through genetic algorithms [5], [6], [9], [20], [22], [29], [31]. However, the computationally intensive nature of GA's often makes them prohibitive for real-world applications. As GA's do not explicitly use gradient information (as backpropagation does), large amounts of time may be spent searching before an acceptable solution is found.

In order to reduce search times, a novel evolutionary search algorithm is used in this study: Population-Based Incremental Learning (PBIL). Although a complete description of PBIL's

derivation and performance compared with other evolutionary algorithms is beyond the scope of this paper, a description of its fundamental mechanisms can be found below. More detailed descriptions of the algorithm and results obtained in comparisons with genetic algorithms and hillclimbing can be found in [4].

A. An Overview of Population-Based Incremental Learning

Population-based incremental learning (PBIL) is a combination of iterative and evolutionary optimization techniques [4]. The algorithm is based upon the mechanisms of a genetic algorithm and the weight update rule of supervised competitive learning [16]. The PBIL algorithm, like standard GA's, relies on discrete evaluations of potential solutions. In this study, each potential solution is a fully specified network. Both the network topology and the connection weights are encoded in the potential solution and evolved in the search process. The PBIL algorithm described in this paper operates on potential solutions defined on a binary alphabet.

One of the drawbacks of using a binary alphabet is that the values of the weights of the encoded network must be discretized to a specified precision. As the solution string lengths used in this study have a fixed size, the number of bits allocated to represent each weight is specified before the algorithm is started. The translation of these bits to weights assumes that the bits encode a base-2 number which specifies the value of the weight within a pre-specified range of possible values. Although methods have been presented to avoid this limitation by dynamically varying the number of bits allocated for representing the weights [22], such schemes were not adopted for this study. We found through empirical testing that overestimating the number of required bits did not hinder performance, although it did increase the search time. More details on the representations will be given with the description of the specific tests.

The object of the PBIL algorithm is to create a real valued probability vector which specifies the probabilities of having a '1' in each bit position of high evaluation solutions. When this probability vector is sampled, it should generate high evaluation vectors with high probability. For example, if a good solution to a problem can be encoded as a string of alternating 0's and 1's, a suitable final probability vector is 0.01, 0.99, 0.01, 0.99, etc. The probability vector can be considered a "prototype" for high evaluation vectors for the function space being explored.

One of the key features in the early portions of genetic optimization is the parallelism inherent in the search; many diverse points are represented in the population of a single generation [7], [17]. In PBIL, the population of a GA is represented in terms of a probability vector. In the probability vector, the most diversity will be found by setting the values of each bit position to 0.5. This specifies that generating a 0 or 1 in each bit position is equally likely. In a manner similar to the training of a competitive learning network, the values in the probability vector are gradually shifted toward the bit values of high evaluation vectors. A simple procedure to accomplish this is described below. The probability update rule, which is based upon the update rule of standard competitive learning, is shown below:

$$probability_{i,t+1} = (probability_{i,t} * (1.0 - \mathrm{LR})) + (\mathrm{LR} * solution_i) \quad (1)$$

where $probability_{i,t}$ is the probability of generating a 1 in bit position i at time step t, $solution_i$ is the value of the ith position in the highest evaluation vector in the current generation, and LR is the learning rate. The learning rate is a user defined parameter, and was set to 0.1 for this study. The *probability* vector and the *solution* vector both have the same length as the encoded solution.

The step which remains to be defined is determining which solution vectors to move toward. These vectors are chosen as follows: a number of potential solution vectors are generated by sampling the probabilities specified in the probability vector. Each of these potential solution vectors is evaluated with respect to the goal function. For this task, the goal function is how well the encoded ANN performs on the training set. This is determined by decoding the solution vector into the topology and weights of the ANN, performing a forward pass through the training samples, and measuring the error of the outputs. The probability vector is pushed toward the generated solution vector with the lowest total error; it is pushed according to the equation described above. In addition, the probability vector is also moved toward the complement of the vector with the highest total error[1]. After the probability vector is updated, a new set of potential solution vectors is produced; these are based upon the updated probability vector, and the cycle is repeated.

In addition to the update rule shown above, a "mutation" operator is used. This is analogous to the mutation operator used in standard genetic algorithms. Mutation is used to prevent the probability vector from converging to extreme values without performing extensive search. In standard genetic algorithms the mutation operator is implemented as a small probability of randomly changing a value in a member of the population. In the PBIL algorithm, the mutation operator affects the probability vector directly; each vector position is shifted in a random direction with a small probability in each iteration. The magnitude of the mutation shift (0.05) is smaller than the learning rate (0.1).

In the implementation used in this study, the population size is kept constant at 30; the population size refers to the number of potential solution vectors which are generated before the probability vector is updated. This is a very small population size in comparison to those often used in other forms of evolutionary search. For a test suite of small neural network evolution problems, [22] has shown that larger sequential populations may not yield sufficiently better results to warrant the associated extra computational penalties. Because of the small size of the population used, and the probabilistic generation of solution vectors, it is possible that a good vector may not be created in each generation. Therefore, in order to avoid moving toward unproductive areas of the search space,

[1] Actually, the probability vector is moved toward the complement of the vector with the lowest evaluation only in the bit positions in which the highest evaluation vector and the lowest evaluation vector differ.

```
***** Initialize Probability Vector *****
for i :=1 to LENGTH do
        P[i] = 0.5;

while (NOT termination condition) do
begin
        ***** Generate Samples *****
        for i :=1 to NUMBER_SAMPLES do
        begin
                sample_vectors[i] := generate_sample_vector_according_to_probabilities (P);
                evaluations[i] := Decode_and_Evaluate_Network (sample_vectors[i]);
        end;

        best_vector := find_vector_with_best_evaluation (sample_vectors, evaluations);
        worst_vector := find_vector_with_worst_evaluation (sample_vectors, evaluations);

        ***** Update Probability Vector towards best network *****
        for i :=1 to LENGTH do
                P[i] := P[i] * (1.0 - LR) + best_vector[i] * (LR);

        ***** Update Probability Vector away from worst network *****
        for i := 1 to LENGTH do
                if (best_vector[i] ≠ worst_vector[i]) then
                        P[i] := P[i] * (1.0 - NEGATIVE_LR) + best_vector[i] * (NEGATIVE_LR);

        ***** Mutate Probability Vector *****
        for i :=1 to LENGTH do
                if (random (0,1) < MUT_PROBABILITY) then
                begin
                        if (random (0,1) > 0.5) then
                                mutate_direction := 1
                        else
                                mutate_direction := 0;
                        P[i] := P[i] * (1.0 - MUT_SHIFT) + mutate_direction * (MUT_SHIFT);
                end
end;
```

USER DEFINED CONSTANTS (Typical Values Used in this Study):
NUMBER_SAMPLES: the number of vectors generated before update of the probability vector (30).
LENGTH: the number of bits in a generated vector (6,684 - 8,912).
LR: the learning rate, how fast to exploit the search performed (0.1).
NEGATIVE_LR: the negative learning rate, how much to learn from negative examples (0.075).
MUT_PROBABILITY: the probability for a mutation occurring in each position (0.02).
MUT_SHIFT: the amount a mutation alters the value in the bit position (0.05).

Fig. 4. The PBIL algorithm for a binary alphabet. The "elitist" selection mechanism is not shown.

the best vector from the previous population is also kept in the current population; it replaces the worst member of the current population. This solution vector is only used in case a better solution vector is not produced in the current generation. In genetic algorithm literature, this technique of preserving the best solution vector from one generation to the next is termed "elitist selection" [8], and is often used in parameter optimization problems to avoid losing good solutions once they are found. Without elitist selection, the search proceeded slower; the final solution produced in the limited number of generations (3 000) did not perform as well as PBIL with elitist selection.

The PBIL algorithm, which is less complex than even a simple GA, very quickly optimizes many of the functions which are used to gauge GA performance. However, it neither uses the crossover (recombination) operator, nor defines operations directly on the members of the population, both of which are common to genetic algorithms. The basic algorithm is shown in Fig. 4. In the set of standard GA benchmark problems on which PBIL has been compared, the solutions found by PBIL were better than those found with the GA, and were discovered with less computational cost [4].

IV. The Experiments

For the experiments presented here, four sets of data were collected. The first set of data was obtained by driving down a partially shaded single lane paved bicycle path, the second

was obtained by driving back along the same path. The third set of data was obtained by driving on a two lane suburban neighborhood street, the fourth was obtained by driving back along this same street. The training for all of the experiments in this paper was done on a total of 1 000 images. In the training set, 500 images from the first road type were used, traveling only in one direction, and 500 images were used of the second road type, again only traveling in a single direction. The remaining sets of images were used in the validation and testing sets. Each of these four sets of images are composed of typical road scenes, and shifts and rotations of the road in the original images.

In these experiments, the input retina size was 15×16 pixels. This is half the resolution on each axis as is normally used by the standard ALVINN networks. This smaller retina size was used to reduce training times in order to increase the number of training trials for each of the algorithms examined. Samples of images at this resolution are given in the next section; many of the important features in the images are retained at this resolution.

A. Using a Single Output Unit

The first set of experiments used a single output unit to represent the steering direction. The activation of the output unit determined how sharply the steering should be to the left or to the right of center. The target activations of the output, in both the single and 30 output experiments, ranged from -1.0 to $+1.0$.

For the experiments attempted in this paper, a *maximal network* is defined. This is a user-defined prototype network which determines the structure and maximum connectivity of the evolved networks. Through evolution, connections may be removed from the maximal network; however, connections which are not specified in the maximal network cannot be added. This network is used as the basis of the solution encodings for PBIL; a similar scheme has been used in [22]. The presence of each connection is determined by the value of an assigned bit in the encoding. The maximal network architecture used for the single output unit experiments presented here has a 15×16 input layer, five unit hidden layer, and a single unit output layer. The maximal network is a fully-connected strictly layered feedforward neural network; all units in layer i receive their inputs from all of the units in layer $i-1$ only, and provide outputs to all of the units in layer $i+1$ only.

As the PBIL algorithm presented here operates on a binary bit string, the network must be encoded into an appropriate form. The encoding was as follows: each connection in the network is determined to be either present or absent by a single bit. If the connection is present, its weight is encoded as a base-2 number, in a fixed number of bits. If the connection is not present, the bits used for encoding the weight remain in the bit-string, although they have no effect on the network.

B. Using 30 Output Units

In the second set of experiments, the maximal network was defined to be the original ALVINN network architecture, with 15×16 input units. As mentioned earlier, the maximal network only specifies the maximum number of connections which can be present in the evolved network; the number of connections ultimately used by the final architecture is determined through the search process. The maximal network architecture used for the 30 output unit experiments presented here has a 15×16 input layer, four unit hidden layer, and 30 unit output layer. This architecture contains one less hidden unit than the networks tested with a single output unit. Only four hidden units were used to match the current ALVINN system's architecture.

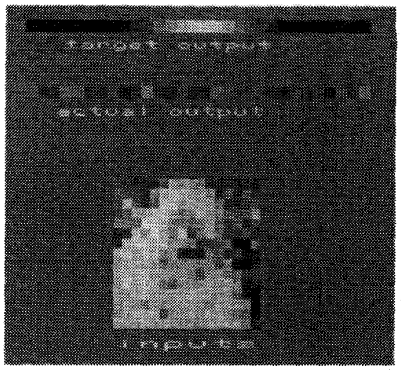


Fig. 5. Input image, target output and network's output before training. The bright pixels represent units with strong positive activation. The dark pixels are units which contain strong negative activation. The target output shows Gaussian activation levels centered around the correct steering direction.

A representative training example is show in Fig. 5. The 15×16 input retina displays a typical road input scene for the network. The target output is also shown. This corresponds to the steering direction the driver of the NAVLAB chose during the test drive made to gather the training images. Also shown is the output of an untrained network. Later in the paper, trained outputs will be shown for comparison.

V. Using Population-Based Incremental Learning

In the evolutionary approach, the need to evaluate each ANN is the source of the largest time penalties. Each ANN must be evaluated to determine which network in the current population has the smallest sum squared error and the largest sum squared error on the training set, as these two examples are used for adjusting the probability vector in the PBIL algorithm. In all the experiments presented in this paper, the training set size was 1 000 images. Because of the large size of the training set, evaluating each network is computationally expensive. One method to reduce the training times is to use a parallel machine. The evaluation of each network is independent from the evaluation of any other network, and can therefore be parallelized very easily. For the algorithm described here, parallelization could lead to dramatic improvements in speed. An alternative training method, which can reduce the computational burden on both serial and parallel machines, is presented below.

Rather than evaluating each network on the entire training set, for each network evaluation a small, randomly selected, portion of the training set is used to measure the network's performance. Although this does not provide an exact in-

dication of the performance of the network on the entire training set, it provides an estimate. The larger the sample size is, the more accurate the estimate becomes. In some of the experiments performed in this study, only 50 out of the 1 000 training examples were used to evaluate each network while still achieving good results. As the computational cost of decoding the solution string into the network is very small in comparison to the time needed to evaluate the network, approximately 20 networks could be evaluated at the cost of evaluating only one if the entire set was used[2]. A somewhat similar technique, which used incrementally larger portions of the training set to train a neural network to recognize speech fragments using backpropagation, was explored by [30].

The drawback of using only a subset of the training set for each network evaluation is the potential noise in each evaluation. The robustness of genetic algorithms in the presence of noise has been examined by [13]. The consequences of this drawback are reduced as the survival of networks throughout a number of consecutive generations will most likely be an indication of their ability to work well on large portions of the training set. In practice, as is shown in the next section, this provides an effective method of reducing the computational burden with little loss in generalization ability. However, if a particular application relied upon the network's capability to memorize the training set, this method would not perform as well. This is shown in more detail in the description of the results given in the next section. All of the results presented in the next sections are the average of at least 10 training sessions.

A. Results With a Single Output Unit

In order to ensure that the test set does not have any effect on the training of the network (including stopping criteria), in all of the experiments presented in this paper, three sets of images were used: 1 000 training images, 100 validation images, and 800 testing images. The *training set* is used to evolve the networks; the network's evaluation is based solely upon the error obtained on the images in this set. To ensure that the network is not memorizing the training set, the errors on a separate set of images, the *validation set*, are used to determine which network will be selected from all of the networks evolved in the run. The network which has the lowest error on the validation set is considered the best network of the run. The generalization ability of the best network is gauged by examining the errors the network produces on the images in the *test set*, which is composed of 800 images. The test set was used only once per run, to gauge the performance of the single network which had the lowest error on the validation set.

For these experiments, one bit was used to represent the presence of each connection, six bits were used to represent the value of each weight, with the ranges of weights between -0.5 and 0.5. The search was allowed to progress 500 generations with 30 networks evaluated per generation. The length of the solution strings were 8 477 bits, which represented a possible 1 211 connections in the network, including the connections from a bias unit. The bias unit has connections to all of the units in the hidden and outputs layers. The bias unit's value is set to $+1.0$ [16].

For a base-line comparison, the maximal network was trained with error backpropagation (BP) with the added features mentioned in the Section II-A. The networks were trained with the 1 000 image training set. The average sum squared error for each image was 0.080 on the 800 image test set. When the networks were trained with PBIL, using only a small randomly sampled portion of the full training set (50 out of 1 000 images) to evaluate each network, the sum squared error was reduced to 0.051 (a 36% decrease from BP) on the test set. If all 1 000 images were used to evaluate each network, the average sum squared error was slightly reduced to 0.048 (a 40% decrease from BP) on the test set. The difference between this result and the one achieved through using only 50 images out of 1 000 for training was found to be not statistically significant. The final networks evolved by both of the PBIL training methods maintained only approximately half of the possible connections of the maximal network.

In Fig. 6, the performances of using the full and sampled training sets are compared. In each figure, both methods of network evaluation are shown. In graph (A), each line represents the best network seen *in the entire run up to the current generation* (averaged over all runs). The four lines correspond to the best network in terms of the training and validation sets, and sampled and full training set methods. Graph (B) shows the progression of the search algorithm. It displays the best network seen *in the population of each generation* (averaged over all runs).[3]

On average, the performances of the networks developed by both training set selection methods do equally well on the validation set. However, using the entire training set decreased the error on the validation set faster, in terms of the number of networks evaluated, than training with a sampled training set. A potential cause for this is the noise in the evaluation of the networks caused by using only a small sample of training examples. As the size of the sample set increases to the entire training set, this problem decreases. Nonetheless, as the size of the sample set increases, the time to perform each network evaluation increases.

B. Results With a 30 Unit Gaussian Encoded Output

As with the single output networks, the tests in this section were performed with the same 1 000 training, 100 validation and 800 image test sets. For these experiments, one bit was used to represent the presence of each connection. Two configurations, which gave approximately equivalent results, were used to represent each weight: the first used six bits, the

[2] With regard to the "elitist selection" mentioned previously: the best vector from the previous generation is re-evaluated in each generation. This may cause fluctuations in its evaluation as the examples on which its evaluation is based are drawn randomly from the full set of examples in the training set.

[3] It should be noted that the validation-set curves in graph B do not reach as low as those in graph A because in individual runs, each network reaches its best performance on the validation-set during different generations. The network which performs the best on the validation set in a generation may be lost in the subsequent generations as it may not have performed the best on the training set. Only the network which performs the best on the training set is explicitly preserved from one generation to the next. Therefore, with averaging over multiple runs, graph A should reach lower as it shows the best network ever seen rather than the best network in the current generation.

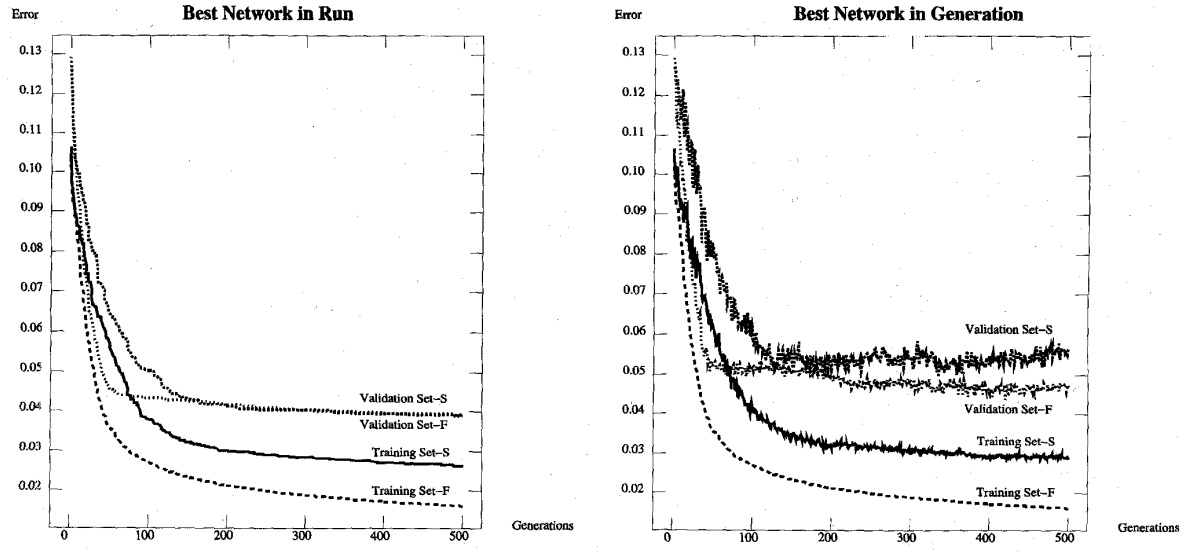


Fig. 6. 1 000 Training, 100 Validation image set. Average error per image is shown. Graph (A) shows the best network seen *in the entire run up to the current generation* (averaged over 10 runs). Graph (B) shows the progression of the search algorithm; it shows the *best network seen in the population of each generation* (averaged over 10 runs). The key (S) means that selection was based only upon a small sample (50 images) of the training set. The (F) means that the performance on the full training set was used in the evaluation of network performance. In order to gauge the relative performances of these two methods, the left graph is more informative than the right. Graph (A) shows the performance of the best network found in the entire run. Graph (B) gives an indication of how the training progresses.

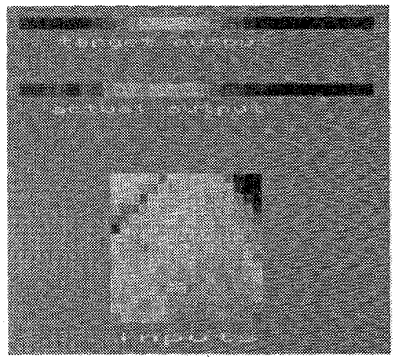
second used seven (this created 7 798 and 8 912 bit solution strings, respectively). The range of weights was between -1.0 and 1.0. The increased range of weight values (in comparison to the single output tests) was determined by examining the weights of the original ALVINN neural network. The range of the weights was larger when 30 outputs were used than when a single output was used. The search was allowed to progress 3 000 generations with 30 networks evaluated per generation. One fifth to one third of the entire training set (200–333 images) was used to evaluate each network. In this section, the networks were evolved to minimize the sum squared error (SSE) between the actual and predicted output vectors. An alternate, task specific, error metric is explored in Section VI. SSE was used in these experiments to provide a comparison with the error metric used by backpropagation.

All of the improvements to the standard backpropagation algorithm which are used to train the ALVINN system were used for obtaining the backpropagation results reported here. These modifications were briefly reviewed in Section II; they are explained in greater detail in [25]. The revised backpropagation algorithm was able to achieve an average sum squared error of 9.40, measured on the 800 image test set. The PBIL algorithm was able to achieve an average error of 8.19, an error reduction of approximately 13%.

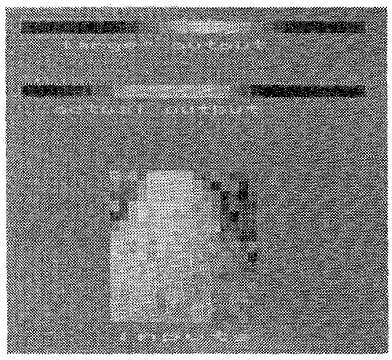
Similarly to the single output tests, the final networks pruned away, on average, approximately half of the possible connections. The drawback of the PBIL approach, in comparison to the backpropagation training method, is the training time; the backpropagation algorithm took only several minutes, on average, to train the networks to its lowest error. The evolutionary approach took over an hour. Nonetheless, the parallelization of the evolutionary algorithm has the potential for large speed improvements. In Fig. 7, a typical sample input and output is given after training by PBIL. A typical sample output attained by standard backpropagation is also shown.

A second test was conducted to determine whether PBIL could perform better if the architecture was pre-specified, and only the weights of the connections were evolved *(PBIL w/Maximal Arch.)*. For this experiment, the maximal network previously described was used. PBIL was only allowed to modify the weights, not the architecture of the ANN. The hope was that if PBIL was constrained to a pre-specified architecture, it may do better because the search space is more constrained. This method should be able to do at least as well, as connections can effectively be eliminated by setting the connection weight to 0.0. Using only the evolution of weights, PBIL was able to find networks which, on average, had an average sum squared error of 7.96; this reflects approximately a 15% improvement over standard backpropagation. This is not significantly different from the results of PBIL evolving both the network architecture and the network weights.

Use of an evolutionary model instead of backpropagation yields a benefit. One of the known strengths of evolutionary techniques is their ability to quickly find regions of high performance. However, after these regions are found, in many evolutionary algorithms, such as genetic algorithms, moves toward optimal points are much slower. Backpropagation has somewhat complementary characteristics; it can quickly find a local optimum from its initial starting point. However, as it is a gradient descent method through the weight space of the network, it cannot perform global search. A natural extension of the model described up to this point is to combine the



(a)



(b)

Fig. 7. (a) The PBIL derived network's output. The actual output for this image has the correct general location as the target output. However, the output is not as smooth as the target. (b) Typical outputs of a network architecture trained with backpropagation. Both figures are taken from the test set.

evolutionary and backpropagation techniques. Several simple methods for combining these learning techniques are described in the next section.

C. Combining Backpropagation and Evolutionary Search

In this section, integrated models of evolutionary and backpropagation learning are presented. These models follow the same fundamental two steps: First, the evolutionary search proceeds as described in the previous section. Second, the backpropagation algorithm is initialized with the results found during evolutionary search. The differences in the approaches described here reside in whether only the evolved network architecture, or only the evolved network weights, or both the evolved architecture and weights are used as a starting point for backpropagation. Similar integration methods are presented in [20]. All of the results presented in this section are summarized in Table I and Table II. The statistical significance of these results is examined after the experiments are presented. All of these experiments are conducted with 30 output unit architectures.

The first method of integration used the network architecture and weights found by PBIL *(PBIL+BP with evolved arch. and evolved weights)*. The network which was used to initialize backpropagation is the network which performed the best in each PBIL run. In less than 3 epochs (passes through the training set in which the network's weights are updated), backpropagation was able to reduce the error to 8.10 (a 13.8% improvement over backpropagation alone).

TABLE I
SUMMARY OF EMPIRICAL RESULTS—SINGLE OUTPUT PBIL COMPARED WITH SINGLE OUTPUT BP AND 30 OUTPUT BP

1 Output	TRAINING METHOD		
	PBIL w/ Sampled Training Set	PBIL w/ Full Training Set	BP
Sum Squared Error	0.051	0.048	0.080
% ERROR DECREASE with respect to 1 output BP (SSE)	36.2	40.0	-
% ERROR DECREASE with respect to 30 output BP (GPPE)	-1.1	1.9	-26.6

The second integration method did not use the weights found by PBIL. In order to test how well backpropagation could do on the network architecture found by PBIL without a good starting point in the weight space, the weights were randomized, and backpropagation attempted *(PBIL + BP with evolved arch. and random weights)*. Each of the ten network architecture developed by PBIL was trained four times with different initial randomized weights, and slightly different parameter settings. The average of the forty runs yielded an average error of 10.13 (this is 7.8% worse than backpropagation alone). This error is higher than any other method examined in this paper.

Two remarks should be made about these results. First, they indicate the possible existence of local minima. When backpropagation is started in a good position in the weight space, as in the first method of integration, it can achieve a much lower error than when started in random locations, as in the second method of integration. Second, the results are suggestive of the fact that PBIL does not evolve networks which will necessarily work well with backpropagation training. Methods for evolving such networks should evaluate each network after backpropagation is applied. Techniques for evolving networks which can learn quickly and accurately have been explored in [19].

In the first two methods of integration, the network architecture found by PBIL was used with backpropagation. In the third method of integration, only the weights found by PBIL were used with the full maximal architecture *(PBIL + BP with maximal arch. and evolved weights)*. The connections which were not present in the architecture evolved by PBIL were given a value of 0.0. This value could be changed through backpropagation training. If BP was initialized with the weights found by PBIL, within three epochs, the error was reduced to 7.99 (a 15% improvement over backpropagation alone).

The fourth, and final, method of integration also used only the weights evolved by PBIL. This method used the weights found by *(PBIL w/maximal arch.)* as the starting point for backpropagation *(PBIL w/maximal arch. + BP with*

TABLE II
SUMMARY OF EMPIRICAL RESULTS—30 OUTPUT PBIL COMPARED WITH 30 OUTPUT BP. TRAINING USED THE SUM SQUARED ERROR METRIC

30 Output Units	TRAINING METHOD						
	PBIL		BP	PBIL AND BACKPROPAGATION			
What PBIL Evolved	Arch & Weights	Weights	-	Arch & Weights	Arch & Weights	Arch & Weights	Weights
How BP was Initialized	-	-	Maximal Arch. Random Weights	Evolved Arch. Evolved Weights	Evolved Arch. Random Weights	Maximal Arch. Evolved Weights	Maximal Arch. Evolved Weights
Sum Squared Error	8.19	7.96	9.40	8.10	10.13	7.99	7.71
% ERROR DECREASE with respect to 30 output BP (SSE)	12.9	15.3	-	13.8	-7.8	15.0	18.0
GAUSSIAN PEAK POSITION ERROR (GPPE)	2.96	2.90	3.35	2.94	3.45	2.96	2.90
% ERROR DECREASE with respect to 30 output BP (GPPE)	11.6	13.4	-	12.2	-3.0	11.6	13.4

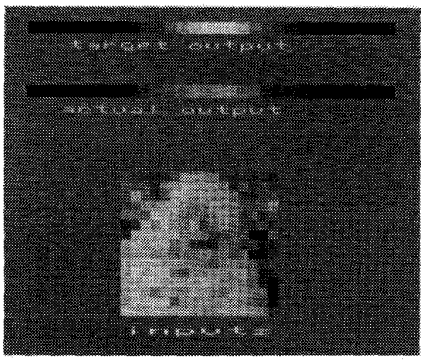


Fig. 8. Result after a final backpropagation pass.

maximal arch. and evolved weights). The difference between this method and the previous one is that in this method, PBIL evolved only the weights of the network, rather than the weights and the network architecture. After three epochs, backpropagation was able to reduce the error to 7.71. This error represents an 18% improvement over backpropagation; it is the lowest error achieved on the sum squared error metric in all of the experiments with 30 output units. A typical sample input/output image after the backpropagation pass is shown in Fig. 8.

D Summary and Significance of Empirical Results

Many results have been presented in this section. In order to judge whether the differences in performance are relevant, the significance for the differences in sum squared error are measured here. For these tests, two sided Mann-Whitney tests, and Kruskal-Wallis ANOVA tests are performed, at the 95% significance level. These tests are nonparametric equivalents of using the standard two-sample pooled t-test, and one-way analysis of variance tests, respectively [28]. The results of the previous sections are summarized in Tables I and II.

For the 30 output units, the worst training method is the *PBIL + BP w/PBIL arch. and random weights* method, which revealed an average SSE error of 10.13. The next worst is standard BP, which revealed an error of 9.40. These errors are significantly different from each other, and from all of the other training methods. The difference between training with PBIL *(in which the architecture and weights are evolved)* and *PBIL w/maximal Architecture (only the weights are evolved)* is not significant, their respective errors are 8.19, and 7.96. The addition of backpropagation training *(PBIL w/maximal arch. + BP w/maximal arch. and evolved weights)* statistically improves the performance over *PBIL (in which the architecture and weights are evolved)*. However, backpropagation does not significantly improve the performance of PBIL when only the weights of the network are evolved. Finally, the difference between *(PBIL + BP w/evolved arch. and evolved weights)* and *(PBIL w/maximal arch. + BP w/maximal arch. and evolved weights)* is significant.

In summary, evolving the architecture and weights of the network does not provide a benefit, in terms of error reduction, over just evolving the weights. Nonetheless, the networks evolved used only approximately half of the connections; therefore, implementation of these networks may be more efficient. In the experiments conducted, evolving the architecture performs a function similar to weight pruning, or free parameter reduction, while the network is constructed. Free parameter reduction is described in detail in [16], [21]. Using backpropagation on the evolved networks improves performance in the cases in which PBIL is used to evolve both the network topology and the weights. Nonetheless, when PBIL is used to evolve only the weights of a network with fixed network topology, using backpropagation did not help significantly in error reduction.

Only very simple integration methods have been presented, more comprehensive integration may reveal improved results. For example, in this study, the evolutionary algorithm was used for weight optimization, with backpropagation applied

at the end. An alternate strategy is to use backpropagation in each generation, so that the EA is used to find a set of weights from which backpropagation can be initialized to perform local optimizations. This changes the search task to finding a good basin of attraction for gradient descent methods rather than an exact set of weights. An approach similar to this was used by [19].

To this point, the effectiveness of evolutionary search techniques has been compared to that of backpropagation using the sum squared error metric. As both algorithms were trained only with this error metric, it is correct to measure their general relative effectiveness by comparing the performances based upon this error. However, in terms of actual performance, domain-specific error metrics are often more indicative of real performance improvements. In this domain, such an error metric is *Gaussian peak position error* (GPPE). This error metric can be used with the 30 unit output architecture; it is a measure of the distance (in output units) between the correct Gaussian peak and the predicted Gaussian peak. It is a linear translation of the steering error. The performances of the training methods gauged by this error metric are also provided in Tables I and II. Since the evolutionary techniques used network evaluations based upon SSE, larger improvements are seen when performance is gauged using SSE than when gauged using GPPE. In the next section, PBIL network training in which network evaluations are based upon the GPPE metric is explored. It should be emphasized that although simple backpropagation could not have directly used the GPPE error metric, evolutionary algorithms can very easily incorporate such information to guide the search.

VI. Using a Task Specific Error Metric

For many problems, the output which is produced by an ANN must be translated into a different form to be used by the specific task. For example, in order to use the 30 output representation, a Gaussian is fit to the outputs, and the peak of the Gaussian is used to determine the predicted steering direction. It is the distance between the predicted and actual peaks of the Gaussian which is crucial to good performance, since this determines the error in the network's steering direction. However, this error measure, GPPE, does not provide an easy mapping of credit or blame to each specific output unit, as is needed for backpropagation. Therefore, sum squared error is often used as the guiding error metric.

In other domains, alternate error metrics have been proposed for backpropagation to better capture the underlying requirements of specific tasks. One such error metric, the Classification Figure of Merit (CFM) error metric has been used for problems such as the 1-of-N classifier. The standard SSE error metric attempts to minimize the difference between each output node and its target activation. The CFM error metric attempts to maximize the difference between the activation of the output node representing the correct classification and the output of all the other nodes which represent incorrect classifications [15]. The CFM error metric concentrates changes on ensuring that the correct classification is made rather than ensuring that the target output is matched exactly.

The GPPE error metric focuses the training toward yielding accurate peak position interpretation of the output vector. Unlike the CFM error metric, however, error cannot easily be assigned to individual output units with the GPPE error metric. Nonetheless, because most evolutionary techniques do not use explicit credit assignment, the GPPE error metric can still be used to guide the evolutionary search. In this section, networks are evolved which explicitly reduce the GPPE rather than the sum squared error; each network is evaluated by its ability to minimize the GPPE error on each image in the training set, without regard to the sum squared error.

The use of the GPPE error metric changes the goal of the search algorithm. Using the SSE error metric, the goal is to reproduce the entire target output vector exactly. Using the GPPE error metric, the goal is to place a larger output activation on the portions of the output vector which correspond to the correct steering direction than those which do not. In fitting a Gaussian to the activations of the output vector, many of the small amounts of noise are ignored. This gives the search procedure the flexibility to not be as precise in large portions of the output vector and still achieve a high score; this is clearly displayed in Fig. 9. The average error using the GPPE error metric was 2.76 (GPPE); this is an 18% improvement over backpropagation networks trained with the SSE error metric (it is statistically significant). These results are an improvement over the previous best networks which were evolved with SSE. The networks evolved with SSE achieved a GPPE error of 2.90, see Table II, (the difference is also statistically significant).

The error measured by GPPE is lower for the evolved networks than the backpropagation networks. However, if the error of these networks is measured in terms of SSE, it is much higher than previous experiments. The average SSE error is 19.3. The large difference in SSE error, in comparison to the other experiments presented before, indicates that doing well in terms of SSE is not a prerequisite for good performance on the error metric of interest, GPPE.

VII. Conclusion

Although the backpropagation algorithm used in this study maintained spatial information about the input retina which the evolutionary algorithm was not given, the evolutionary approach performed better, on average, than backpropagation. The largest improvements occurred in the one output networks (36–40%). This study, and previous studies which have used alternative single output network architectures trained with backpropagation, have shown the performance of one output unit to be much worse than that of a distributed thirty output unit network [25]. Through the evolutionary approach, the single output representation is equivalent in performance to the thirty unit backpropagation network, measured in terms of steering error. In the experiments in which the thirty output unit networks were evolved, the evolved networks performed 18% better than the networks trained with error backpropagation. All of the network topologies which are evolved in this study are efficient; they use approximately only half of the total number of possible connections.

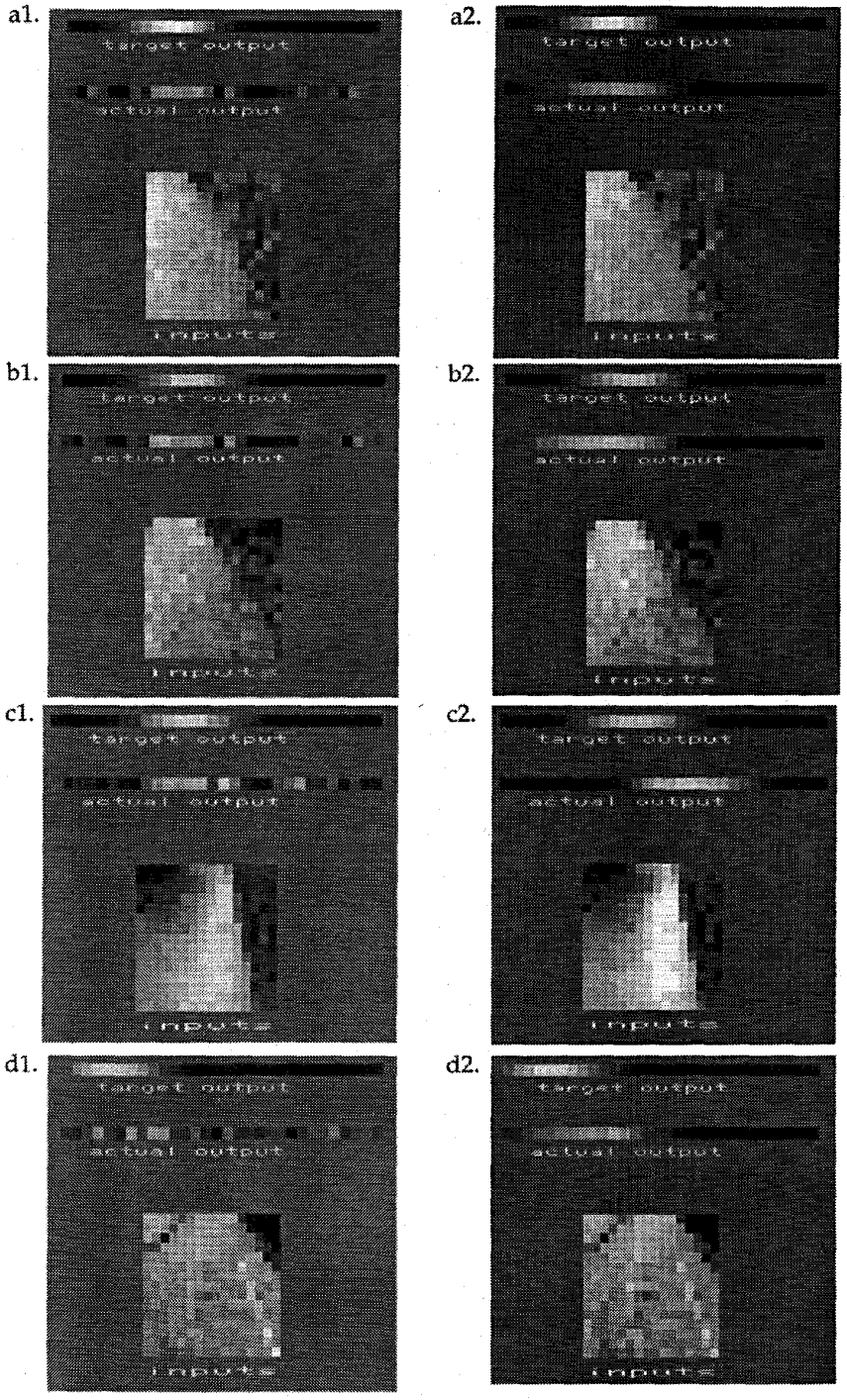


Fig. 9. The left column shows sample input and output pairs using the GPPE error metric. Images taken from test set. Errors in image a1: 0.53, b1: 1.16, c1: 1.71, d1: 2.41. In the right column, sample input and outputs pairs using backpropagation with the standard SSE error metric are shown. These images have the following GPPE errors: a2: 0.49, b2: 1.70, c2: 5.12, d2: 3.83. Images were chosen to show a range of GPPE errors.

Evolutionary search procedures have the ability to perform global search. Backpropagation, on the other hand, provides the ability to perform local optimizations. In these experiments, the EA was used until a good network was found. Using backpropagation to perform local optimizations on the evolved networks yielded slightly improved results over either method alone.

In this study, the GPPE and SSE error metrics were used to guide the evolutionary search. Using the GPPE metric did not produce as smooth output vectors as the SSE metric. However,

in terms of steering error, networks evolved with the GPPE metric performed better than networks evolved with SSE alone. When the networks evolved with SSE used backpropagation for local optimizations, the results were comparable to the networks evolved with GPPE alone.

The drawback of using an evolutionary approach is the severe computational penalties. For example, in training the networks with thirty output units, the backpropagation method was able to achieve its minimum in several minutes, the evolutionary approach took over an hour. Nonetheless, as the evaluation of networks on the training set is the largest computational burden, parallelization of this will lead to improved training times. In determining whether or not an evolutionary approach is appropriate for a particular application, the conflicting needs for accuracy and speed must be taken into careful consideration.

This paper has presented several techniques for increasing the efficiency of evolutionary procedures for training artificial neural networks. The first is the evolutionary procedure used—population based incremental learning; this has been shown to be more effective and efficient than standard GA's in a variety of optimization problems [4]. The second is the evaluation of each network on a small subset of the original training set instead of on the entire training set. As mentioned before, since the majority of the time in the evolutionary search procedure is spent evaluating each network's effectiveness on the training set, reducing the time for evaluation has a tremendous impact on the overall speed of learning.

VIII. Future Directions

One of the first areas in which this study should be directed is to examine which features the hidden units have developed. Pomerleau and Touretzky [26] have examined the features developed in the ALVINN networks. It will be interesting to determine whether the features developed through evolution are similar to those developed using only backpropagation. It is suspected that because the backpropagation algorithm used in ALVINN uses a high neighbor smoothing rate, that perhaps if a similar mechanism was incorporated into the evolutionary algorithm, the features developed may be similar to those found in the standard ALVINN networks.

In many of the previous studies in which artificial neural networks were evolved, either the entire network structure was evolved from only the inputs and outputs specified, or only the weights were evolved once the network was entirely prespecified. The method presented in this paper, which is similar to the one chosen by [22], is a hybrid between these two extremes. This hybrid approach allows any *a priori* information regarding the network topology to be easily incorporated into the search. Nonetheless, a more automatic specification of the network and scaling of weight ranges would be desirable. For example, the maximal network limits the architecture of the network to a pre-specified form. A direction for future work is allowing any arbitrary architecture to be evolved, as in [2]. The need for automatic specification of the network architecture and weights must be balanced by the larger search times which genetic or PBIL methods will require.

The version of population-based incremental learning which was used in this study was very simple. More advanced versions of the algorithm may yield better results. For example, although the cardinality of the networks encoding was two, the PBIL algorithm can also work on alphabets of larger cardinality. Either using a larger cardinality alphabet or using real valued features may improve the performance of the algorithm. Fogel et al., have studied evolving neural networks with real values rather than binary encoded values, and have achieved good results [10].

In the experiments presented here, a fraction of the entire training set was used. However, no method for determining the correct fraction of the training set to use was presented. If too large a portion is used, search can be slowed down tremendously. If too small a portion is used, the noise in the evaluation of each network may lead the search algorithm away from finding networks which perform well on the entire training set. As using only a few samples from the training set has the potential to greatly reduce the amount of time used for developing networks with good generalization capabilities, more rigorous statistical techniques should be used to define the sample size needed for network evaluations.

The long-term future goal, of which this project is a part, is to collect a pool of the evolved networks which can be installed into the NAVLAB. However, before this can be done, several issues need to be resolved, such as determining the best method of using a pool of networks. One alternative is a system like MANIAC, in which a meta-level network controls which expert networks (networks trained on specific road types) will contribute to the final outcome [18]. Other alternatives are systems which use reliability estimates of each network to determine which network should control the steering direction [25]. The second issue which needs to be addressed is that of training networks with more than a single road type. The desire to use a small number of expert networks must be weighed against the potential performance degradation of networks trained on more than a single road type. The transition to using evolved networks, whether they are eventually evolved with single or multiple road types, will not be difficult. The evolved networks can use the same inputs and outputs as the ALVINN networks. Their outputs are translated into steering commands in exactly the same manner as the ALVINN networks already in use. The results shown here appear promising in their error reduction; nonetheless, only actual use will determine their true efficacy.

Acknowledgment

The author is grateful to D. Pomerleau for his extended discussions of the ideas contained within this report. C. Thorpe provided invaluable discussions and encouragement. Thanks are also extended to M. Dorigo and to the reviewers for their suggestions. T. Jochem provided the sample images from which all training and testing was conducted. Finally, thanks are also due to the members of the CMU-Vision and Autonomous Systems Center for enduring the computational costs associated with this project.

REFERENCES

[1] D. H. Ackley, *A Connectionist Machine for Genetic Hillclimbing*. Norwell, MA: Kluwer, 1987.

[2] P. Angeline, G. Saunders, and J. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Trans. Neural Networks*, vol. 5, no. 1, pp. 54–64, 1994.

[3] T. Baeck and H. P. Schwefel, "An overview of evolutionary algorithms for parameter optimization," *Evolutionary Comp.*, vol. 1, no. 1, pp. 1–24, 1993.

[4] S. Baluja, "Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning," Tech. Rep. CMU-CS-94-163, Dept. Comp. Sci., Carnegie-Mellon Univ., 1994.

[5] R. D. Beer and J. C. Gallagher, "Evolving dynamical neural networks for adaptive behavior," *Adapt. Beh.*, vol. 1, no. 1, pp. 92–122, 1992.

[6] R. Belew, "Interposing an ontogenic model between genetic algorithms and neural networks," *Advances in Neural Information Processing Systems 5*, S. J. Hanson, J. D. Cowan, and C. L. Giles, Eds. San Mateo, CA: Morgan Kaufmann, 1993, pp. 99–106.

[7] A. Bertoni, and M. Dorigo, "Implicit parallelism in genetic algorithms," *Artif. Intell.*, vol. 61, no. 2, pp. 307–314, 1993.

[8] K. De Jong, "An analysis of the behavior of a class of genetic adaptive systems," *Doctoral Dissertation*, Dept. Comp. Comm. Sci., Univ. of Michigan, Ann Arbor,. 1975.

[9] D. Floreano and F. Mondada, "autonomous and self-sufficient: Emergent homing behaviors in a mobile robot," this issue, pp. 396–407.

[10] D. B. Fogel, L. J. Fogel, and W. Porto, "Evolving neural networks," *Biolog. Cybern.*, vol. 63, pp. 487–493, 1990.

[11] D. B. Fogel, "An introduction to simulated evolutionary optimization," *IEEE Trans. Neural Networks*, vol. 5, no. 1, pp. 3–13, 1994.

[12] D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[13] J. J. Grefenstette and J. M. Fitzpatrick, "Genetic algorithms in noisy environments," in *Machine Learning*. Norwell, MA: Kluwer, 1988, vol. 3, nos. 2/3, pp. 101–120.

[14] F. Gruau, "Genetic synthesis of modular neural networks," in *Proc. Fifth Int. Conf. on Genetic Algorithms*, S. Forrest, Ed., 1993, pp. 318–324.

[15] J. B. Hampshire and A. H. Waibel, "A novel objective function for improved phoneme recognition using time-delay neural networks," Tech. Rep. CMU-CS-89-118, Dept. Comp. Sci., Carnegie-Mellon University, 1989.

[16] J. Hertz, J. A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*. Reading, MA: Addison-Wesley, 1993.

[17] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: The Univ. of Michigan Press.

[18] T. M. Jochem, D. A. Pomerleau, and C. E. Thorpe, "MANIAC: A next generation neurally based autonomous road follower," in *Proc. Int. Conf. on Intelligent Autonomous Systems (IAS-3)*, F. C. A Groen, S. Hirose, and C. E. Thorpe, Eds., 1993, pp. 592–599.

[19] R. Keesing and D. Stork "Evolution and learning in neural networks: The number and distribution of learning trials affect the rate of evolution," *Advances in Neural Information Processing Systems 3*, R. P. Lippman, J. E. Moody, and D. S. Touretzky, Eds. San Mateo, CA: Morgan Kaufmann, pp. 804–810, 1991.

[20] H. Kitano, "Empirical studies on the utility of genetic algorithms for training and designing neural networks," Tech. Rep. CMU-CMT-90-120, Center for Machine Translation, Carnegie Mellon Univ., 1990.

[21] Y. Le Cun, J. S. Denker, and S. A. Solla, "Optimal brain damage," *Advances in Neural Information Processing Systems II*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, pp. 598–605.

[22] V. Maniezzo, "Genetic evolution of the topology and weight distribution of neural networks," *IEEE Trans. Neural Networks*, vol. 5, no. 1, pp. 39–53, Jan. 1994.

[23] D. A. Pomerleau, "Efficient training of artificial neural networks for autonomous navigation," *Neural Comp.*, vol. 3, no. 1, pp. 88–97, 1991.

[24] D. A. Pomerleau, "Progress in neural network-based vision for autonomous robot driving," in *Proc. 1992 Intelligent Vehicles Conf.*, I. Masaki, Ed., 1992, pp. 391–396.

[25] D. A. Pomerleau, *Neural Network Perception for Mobile Robot Guidance*. Norwell, MA: Kluwer, 1993.

[26] D. A. Pomerleau and D. S. Touretzky, "Analysis of feature detectors learned by a neural network autonomous driving system," in *Proc. Int. Conf. on Intelligent Autonomous Systems (IAS-3)*, F. C. A. Groen, S. Hirose, C. E. Thorpe, Eds., 1993, pp. 572–581.

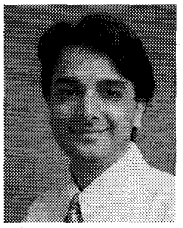
[27] D. A. Pomerleau, C. Thorpe, D. Langer, J. K. Rosenblatt, and R. Sukthankar, "AVCS research at Carnegie Mellon University," in *Proc. Intelligent Vehicle Highway Systems America 1994 Annual Meeting*, 1994, pp. 257–262.

[28] B. F. Ryan, B. L. Joiner, and T. A. Ryan, *Minitab Handbook*. Boston, MA: PWS-Kent, 1985.

[29] J. D. Schaffer, D. Whitley, and L. J. Eshelman, "Combinations of genetic algorithms and neural networks: A survey of the state of the art," *COGANN-92: Int. Workshop on Combinations of Genetic Algorithms and Neural Networks*, D. Whitley and J. D. Schaffer, Eds., 1992, pp. 1–37.

[30] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. Lang "Phoneme recognition using time-delay neural networks," *Readings in Speech Recognition*, A. Waibel and K. F. Lee, Eds. San Mateo, CA: Morgan Kaufmann 1990, pp. 393–404.

[31] B. M. Yamauchi and R. D. Beer, "Sequential behavior and learning in evolved dynamical neural networks," *Adapt. Beh.*, vol. 2, no. 3, pp. 219–246, 1994.



Shumeet Baluja was born in New Delhi, India, on February 17, 1971. He received the B.S. degree in computer science from the University of Virginia, Charlottesville, in 1992. He is currently pursuing the Ph.D. degree in computer science at Carnegie Mellon University, Pittsburgh, PA.

His research interests include artificial neural networks and their applications, mechanisms for selective attention in vision, learning in visual domains, and evolutionary algorithms.