Chapter 1

Shumeet Baluja
School of Computer Science
Carnegie Mellon University

baluja@cs.cmu.edu

Artificial Neural Network Evolution: Learning to Steer a Land Vehicle

Abstract
This chapter presents an evolutionary method for creating an artificial neural network based controller for an autonomous land vehicle. Previous studies which have used evolutionary procedures to evolve artificial neural networks have been constrained to small problems by extremely high computational costs. In this chapter, methods for reducing the computational burden are explored. Previous connectionist based approaches to this task are discussed. The evolutionary algorithrm used in this study, Population-Based Incremental Learning (PBIL), is a variant of the traditional genetic algorithm. It is described in detail in this chapter. The results indicate that the evolutionary algorithm is able to generalize to unseen situations better than the standard method of error backpropagation; an improvement of approximately 18% is achieved on this task. The networks evolved are efficient; they use only approximately half of the possible connections. However, the evolutionary algorithm may require considerably more computational resources on large problems.

0-8493-2529-3/95/$0.00 + $.50

1.1 Overview

In this chapter, 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 artificial neural networks (ANNs) have concentrated on relatively small problems. This chapter presents a study of evolutionary optimization 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 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 chapter. Methods for avoiding the high computational costs 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 realtime robotic environments, is not addressed in this chapter. The benefit of evolutionary algorithms lies in their ability to perform global search; they provide a mechanism which is more resistant to local optima than standard backpropagation. In determining whether an evolutionary approach is appropriate for a particular application, the conflicting needs for accuracy and speed must be taken into careful consideration.

The next section very briefly reviews the fundamental concepts of ANNs. This material will be familiar to the reader who has had an introduction to ANNs. Section 1.3 provides an overview of the currently used artificial neural network based steering controller for the NAVLAB, named ALVINN (Autonomous Land Vehicle in a Neural Network) [16]. Section 1.4 gives the details of the evolutionary algorithm used in this study to evolve a neuro-controller; Population-Based Incremental Learning [4]. Section 1.5 gives the details of the task. Section 1.6 gives the implementation and results. Finally, Sections 1.7 and 1.8 close the chapter with conclusions and suggestions for future research.

1.2 Introduction to Artificial Neural Networks

An Artificial Neural Network (ANN) is composed of many small computing units. Each of these units is loosely based upon the design of a single biological neuron. The models most commonly used are far simpler than their biological counterparts. The key features of each of these simulated neurons are the inputs, the activation function, and the outputs. A model of a simple neuron is shown in Figure 1.1. The inputs to each neuron are multiplied by connection weights giving a net total input. This net input is passed through a non-linear activation function, typically the sigmoid or hyperbolic tangent function, which maps the infinitely ranging (in theory) net input to a value between set limits. For the sigmoidal activation function, input values will be mapped to a point in (0,1) and for the hyperbolic tangent activation function, the input will be mapped to a value in (-1,1). Once the resultant value is computed, it can either be interpreted as the output of the network, or used as input to another neuron. In the study presented in this chapter, hyperbolic tangent activations were used.

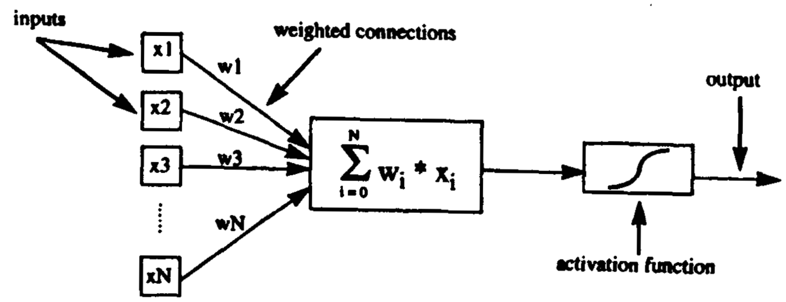

Figure 1.1: The artificial neuron works as follows: the summation of the incoming (weights * activation) values is put through the activation function in the neuron. In the above shown case, this is a sigmoid. The output of the neuron, which can be fed to other neurons, is the value returned from the activation function. The x's can either be other neurons or inputs from the outside world.

Artificial neural networks are generally composed of many of the units shown in Figure 1.1, as shown in Figure 1.2. For a neuron to return a particular response for a given set of inputs, the weights of the connections can be modified. "Training" a neural network refers to modifying the weights of the connections to produce the individual output vector associated with each input vector.

A simple ANN is composed of three layers, the input layer, the hidden layer and the output layer. Between the layers of units are connections containing weights. These weights serve to propagate signals through the network. (See Figure 1.2.) Typically, the network is trained using a technique which can be thought of as gradient descent in the connection weight space. Once the network has been trained, given any set of inputs and outputs which are sufficiently similar to those on which it was trained, it will be able to reproduce the associated outputs by propagating the input signal forward through each connection until the output layer is reached.

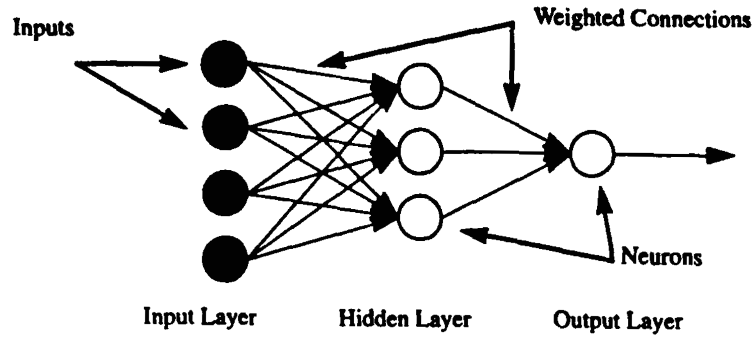

Figure 1.2: A fully connected three layer ANN is shown. Each of the connections can change its weight independently during training.

In order to find the weights which produce correct outputs for given inputs, the most commonly used method for weight modification is error backpropagation. Backpropagation is simply explained in Abu-Mostafa's paper "Information Theory, Complexity and Neural Networks"[1]:

> ...the algorithm [backpropagation] operates on a network with a fixed architecture by changing the weights, in small amounts, each time an example $y_i = f(x_i)$ [where y is the desired output pattern, and x is the input pattern] is received. The changes are made to make the response of the network to x_i closer to the desired output, y_i. This is done by gradient descent, and each iteration is simply an error signal propagating backwards in the network in a way similar to the input that propagates forward to the output. This fortunate property simplifies the computation significantly. However, the algorithm suffers from the typical problems of gradient descent, it is often slow, and gets stuck in local minima.

If ANNs are not overtrained, after training, they should be able to generalize to sufficiently similar input patterns which have not yet been encountered. Although the output may not be exactly what is desired, it should not be a catastrophic failure either, as would be the case with many non-learning techniques. Therefore, in training the ANN, it is important to get a diverse sample group which gives a good representation of the input data which might be seen by the network during simulation. A much more comprehensive tutorial of artificial neural networks can be found in [12].

1.3. Introduction to ALVINN

ALVINN is an artificial neural network based perception system which learns to control Carnegie Mellon's NAVLAB vehicles by watching a person drive, see Figure 1.3. ALVINN's architecture consists of a single hidden layer backpropagation network. The input layer of the network is a 30x32 unit two dimensional "retina" which receives input from the vehicle's video camera, see Figure 1.4. 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 output unit with the highest activation represents 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. The backpropagation algorithm alters the strengths of connections between the units so that the network produces the appropriate steering response when presented with a video image of the road ahead of the vehicle.

Because 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 bike paths, two lane suburban neighborhood streets, and lined divided highways. In this last domain, ALVINN has successfully driven autonomously at speeds of up to 55 m.p.h., and for distances of over 90 miles on a highway north of Pittsburgh, Pennsylvania.

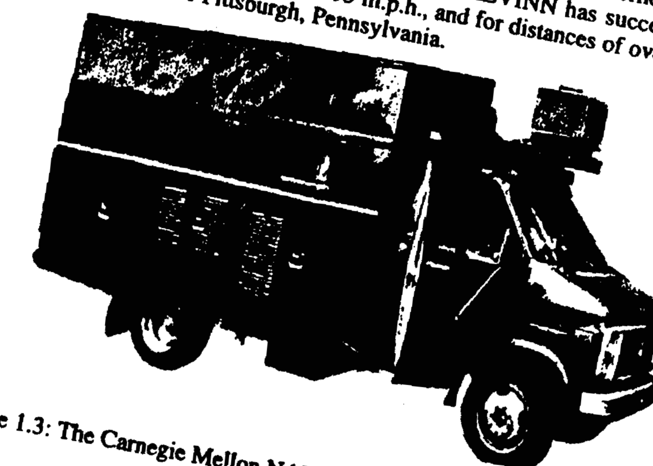

Figure 1.3: The Carnegie Mellon NAVLAB Autonomous Navigation testbed.

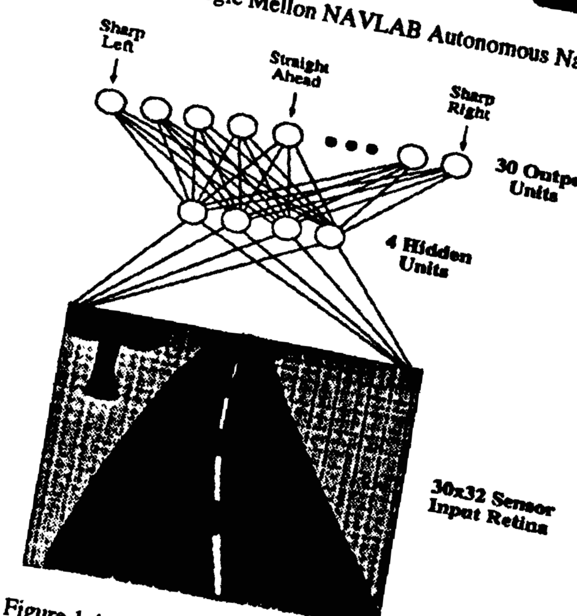

Figure 1.4: The ALVINN neural network architecture.

The performance of the ALVINN system has been extensively analyzed by Pomerleau [16][17][18]. Throughout testing, various 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 [15][16]. In the study presented here, the output representation examined is the one currently used in the ALVINN system, a distributed representation of 30 units.

1.3.1 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 correct steering direction is the steering direction the human driver of the NAVLAB has chosen. The weights in the network are altered using the backpropagation algorithm so that the network's output more closely corresponds to the target output. 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 are spatially close to each other, in terms of their connections to the units in the input retina, to similar values. This is a method of preserving spatial information in the context of the backpropagation algorithm.

In its current implementation, ALVINN is trained to produce a Gaussian distribution of activation centered around the appropriate steering direction. However, this steering direction may fall between the directions represented by two output units. A Gaussian approximation is used to interpolate the correct output activation levels of each output unit. Using the Gaussian approximations, the desired output activation levels for the units successively farther to the left and the right of the correct steering direction will fall off rapidly on either side of the two most active units. A representative training example is shown below, in Figure 1.5. The 15x16 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 chapter, trained outputs will be shown for comparison.

One of the problems associated with this training 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 slightly 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. The network is trained on the original and shifted images.

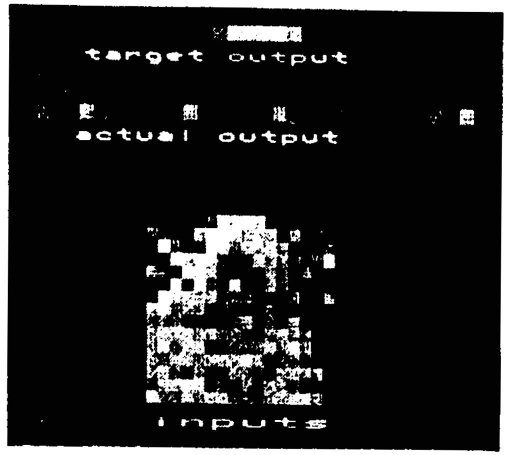

Figure 1.5: Input image, target and actual outputs before training.

1.4 The Evolutionary Approach

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 predefined 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 weights space, and can therefore get stuck in local minima. Evolutionary algorithms (EAs) are methods of global search, and are less susceptible to local minima. Finding the appropriate set of weights in a neural network can be formulated as parameter optimization problem to which EAs can be applied in a straightforward manner. A much more comprehensive overview of evolutionary algorithms, and their applications to parameter optimization tasks, can be found in [3] [9].

The second method for applying EAs endeavors 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 backpropagation to train the networks specified by the search, or can simultaneously be found while searching for the network topology. The method explored in this chapter is a variant of the latter approach, and will be described in much greater detail in the following sections. The advantage to this method is that if there is very little knowledge of the structure of the problem, and therefore no knowledge, other than the number of inputs and outputs needs that need to be incorporated into the network, the structure of the network does not need to be predefined in detail.

Given the possibility of backpropagation falling into a local minima, and the potential lack of knowledge regarding the appropriate neural network architecture to use, using EAs appears to be a good alternative. However, the largest drawback of EAs, and the one which has made them prohibitive for many "real world" learning applications, is their enormous computational burden. As EAs do not explicitly use gradient information (as backpropagation does), large amounts of time may be spent searching before an acceptable solution is found.

Previous work has been done to measure the feasibility of evolutionary approaches on standard neural network benchmark problems, such as the encoder problem, and exclusive-or (XOR) problems, etc. More complicated problems have also been attempted, such as the control of an animat which learns to play soccer, given a small set of features about the environment such as the ball position, the status of the ball, etc. with good results [14]. Other work, which has concentrated on solving a "search and collection task" of simulated ants has used the evolution of recurrent neural networks, with evolutionary programming, again with successful results [2].

Many of the studies which have used evolution as the principle learning paradigm of training artificial neural networks have often modelled evolution through genetic algorithms [6][10][14]. However, genetic algorithms are very computationally expensive for large problems. In order to reduce the search times, a novel evolutionary search algorithm is used in this study. The algorithm, Population Based Incremental Learning (PBIL), is based upon the mechanisms of a generational genetic algorithm and the weight update rule of supervised competitive learning [12]. Although a complete description of its derivation and its performance compared with other evolutionary algorithms is beyond the scope of this chapter, 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].

1.4.1 Population-Based Incremental Learning

PBIL is an evolutionary search algorithm based upon the mechanisms of a generational genetic algorithm and supervised competitive learning. The PBIL algorithm, like standard genetic algorithms, does not use derivative information; rather, it relies on discrete evaluations of potential solutions. In this study, each potential solution is a fully specified network; both the topology and the connection weights can be encoded in the potential solution and evolved in the search process. The PBIL algorithm described in this chapter operates on potential solutions defined in a binary alphabet. The exact encodings of the networks will be described in the next section.

The fundamental goal of the PBIL algorithm is to create a real-valued probability vector which specifies the probability of having a '1' in each bit position of the potential solution. The probabilities are created to ensure that potential solutions, from which the individual bits are drawn with the probabilities specified in the probability vector, have good evaluations with a high probability. The probability vector can be considered a "prototype" for high evaluation vectors for the function space being explored.

A very basic observance of genetic algorithm behavior provides the fundamental guidelines for the performance of PBIL. 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. In representing the population of a GA in terms of a probability vector, the most diversity will be found in setting the probabilities 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 towards 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.

$$probability_i = (probability_i \times (1.0 - LR)) + (LR \times solutionVector_i)$$

$probability_i$ is the probability of generating a 1 in bit position i.
$solutionVector_i$ is the value of the ith position in the high evaluation vector.
LR is the learning rate (defined by the user).

The *probability vector and the solutionVector* are both the length of the encoded solution.

The step which remains to be defined is determining which solution vectors to move towards. The vectors are chosen as follows: a number of potential solution vectors are generated by sampling from the probabilities specified in the current 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 sum squared error of the outputs. The probability vector is pushed towards the generated solution vector with the best evaluation: the network with the lowest sum squared error. 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 continued.

During the probability vector update, the probability vector is also moved towards the complement of the vector with the lowest evaluation. However, this move is not made in all of the bit positions. The probability vector is moved towards 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.

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 shift is small in comparison to the learning rate.

The probability vector is adjusted to represent the current highest evaluation vector. As values in the bit positions become more consistent between highest evaluation vectors produced in subsequent generations, the probabilities of generating the value in the bit position increases. The probability vector has two functions, the first is to be a prototype for high evaluations vectors, and the second is to guide the search from which it is further refined.

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. Because of the small size 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 towards unproductive areas of the search space, the best vector from the previous population is also kept in the current population. This solution vector is only used in case a better evaluation 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," and is often used in parameter optimization problems to avoid losing good solutions, by random chance, once they are found.

```
P <-- initialize probability vector. (Each position = 0.5)
loop # GENERATIONS
  ***** Generate Samples *****
  i ← loop #SAMPLES
    sample_i ←  generate  sample  vector  according  to
                                        probabilities in P
    evaluation_i ← Decode_Network_and_Perform_Forward_Pass
                                        (sample_i)
  best ← find vector corresponding to best evaluation
  worst ← find vector corresponding to worst evaluation
  ***** Update Probability Vector towards best network *****
  I ← loop #LENGTH
    P_I ← P_I * (1.0 - LR) + best_I * (LR)
  ***** Update Probability Vector away from worst network *****
  I ← loop #LENGTH
    if (best_I ≠ worst_I) P_I ← P_I * (1.0 - NEGATIVE-LR) + best_I
    * (NEGATIVE-LR)
  ***** Mutate Probability Vector *****
  I ← loop #LENGTH
    if (random (0,1) < MUT_PROBABILITY)
      P_I ← P_I (1.0 - MUT_SHIFT) + random (0.0 or 1.0) *
      (MUT_SHIFT)
USER DEFINED CONSTANTS:
GENERATIONS: how many iterations the algorithm is allowed to
   continue.
SAMPLES: the number of vectors generated before update of the
   probability vector
LENGTH: the number of bits in a generated vector
MUT_PROBABILITY: the probability for a mutation occurring in each
   position
MUT_SHIFT: the amount a mutation alters the value in the bit
   position
LR: the learning rate, how fast to exploit the search performed
   so far.
NEGATIVE-LR: the negative learning rate, how much to learn from
   negative examples.
```


Figure 1.6: The general PBIL algorithm for a binary alphabet. The "elitist" selection mechanism is not shown.

A complete discussion of the merits and drawbacks of this algorithm, as compared to a standard genetic algorithm, is beyond the scope of this chapter. This algorithm, which is far less complex than even a simple genetic algorithm, very quickly optimizes many of the functions which are used to gauge genetic algorithm performance. It does not, however, use the crossover (recombination) operators, or define operations directly on the members of the population, both of which are common to genetic algorithms. The basic algorithm is shown in Figure 1.6. In the set of standard GA benchmark problems on which PBIL has been compared, the resulting solutions found by PBIL are better than those found with the genetic algorithm, and are discovered with far less computational cost [4].

1.5 Task Specifics

The central task explored in this chapter is to develop ANNs for control of an autonomous land vehicle. The specific goal is to develop ANNs which are able to generalize beyond their training set. The motivation for this task is the current project at Carnegie Mellon University to create a pool of pre-trained ALVINN networks, each of which is trained on different road types, under different conditions, etc., from which the appropriate network can be chosen to achieve the most accurate steering direction. Special purpose hardware is currently being designed to allow pre-trained neural networks to control the steering direction. The hardware design does not support modification of the weights; therefore on-line training of the networks is not possible. The goal of this project is to create pre-trained networks which perform well in the encountered situations. This task differs in several ways from the standard ALVINN task. The first difference is that a large degree of good generalization, while important, is not crucial in standard ALVINN tests, as ALVINN is frequently trained to adapt to changing conditions. Secondly, this task does not have to be done on-line. In the standard ALVINN task, training speed is crucial, as it must be able to adapt "on-the-fly" to changing lighting conditions, changing road-types, and changing weather conditions, etc. In this task, the on-line training is not required, as the network pool is trained before any of the networks are used.

For the experiments presented here, four sets of data were collected. Two sets of data were obtained by driving back and forth on a partially shaded single lane paved bicycle path. The other two sets of data were obtained by driving back and forth on a a two lane suburban neighborhood street. The training for all of the experiments in this chapter was done on a total of 1000 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. Each of these four sets of images are composed of images of typical road scenes, and images which contain shifts and rotations of the road in the original images. More details of how these shifts and rotations are performed, and their use in training standard ALVINN networks, can be found in [16].

As mentioned before, the PBIL algorithm used in this study operates on binary strings. 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 are also of fixed size, the number of bits allocated to represent each weight are pre-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. It has been found through empirical testing that overestimating the number of required bits did not hinder performance, although it did increase the search time.

The encodings of the networks into binary strings was as follows: each connection in the network is determined to be either present or absent by a single bit. The weight of the connection, if it is present, is determined by a pre-specified number of bits, and is encoded as a base-2 number. See Figure 1.7 for an example. Although in these experiments the number of bits to represent a connection weight were prespecified, this is not a requirement for evolutionary procedures. An alternative method which avoids this limitation, in which the granularity of detail is also evolved in the search process, is described in [14].

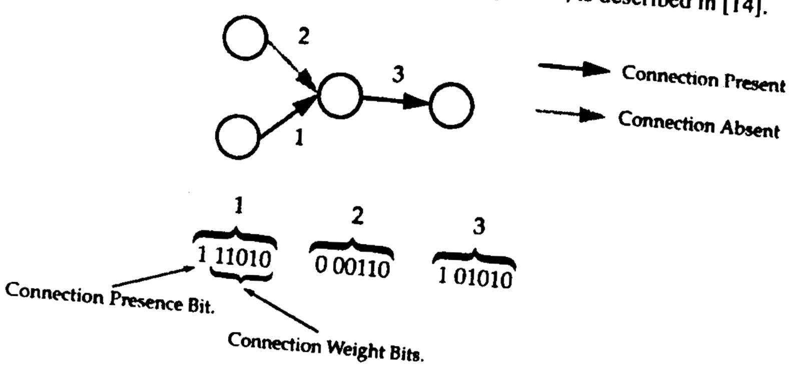

Figure 1.7: Network Encoding into binary form. In this example, the weight of each connection is represented with 5 bits. The presence of each connection is determined by the value of an additional bit.

In the experiments attempted in this chapter, the *maximal network*, a network which specifies the maximum number of feed forward connections allowed (this is user defined), is used as the basis of the solution encodings used in PBIL. This network employed for this study is similar to the one shown in Figure 1.4, with 30 output units, 5 hidden units, and a 15x16 input retina. This architecture is the maximal network, the number of connections ultimately used by the final architecture is determined through the search process.

1.6 Implementation and Results

In an 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. As mentioned before, the evaluation of each network is proportional to the sum squared error between the target and predicted outputs for each image in the training set. In the experiments presented in this chapter, the training set size was 1000 images. With this size training set, evaluating each network is very computationally expensive. A training method designed to reduce the computational burden 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 indication of the performance of the network on the entire training set, it provides an estimate. The larger the sample size, the more accurate the estimate. The drawback of using only a subset of the training set for each evaluation is the potential noise in each evaluation. However, 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, this provides an effective method of reducing the computational burden, with little loss in generalization ability. However, this method should not be used if generalization is not crucial. If the network's capability of memorizing the training set is important, this method does not perform as well [5].

Tests were performed with 1000 training, 100 validation and 800 image test sets. Training was only guided by the errors the network accrued on the training set. The validation set was used to gauge which network should be chosen at the end of the search to test the generalization ability of the network. The generalization ability is gauged on the 800 image test set. The test set was only used once per run, to gauge the performance of the single network which had the lowest average error on the validation set. All of the results are reported in terms of the error on the test set.

For these experiments, 7 bits were used to represent each weight, with the ranges of weights between -1.0 and 1.0. The search was allowed to progress 3000 generations with 30 networks evaluated per generation. To avoid problems with noise in the network's evaluation, one fifth to one third of the entire training set (200-333 images) were used to evaluate each network.

Other tests have been performed with networks which have only a single output unit [5]. The steering direction is determined by the activation of the output unit. In those tests, the same training, validation, and testing sets were used. Results revealed that the use of samples sizes as small as 50 images for each network's evaluation led to approximately equivalent performance, in terms of generalization ability, with using the entire 1000 image training set for each evaluation [5]. However, larger sample sizes are needed for networks which use the 30 output distributed steering direction representation described in this chapter.

The encoding of the network used a bit string which is longer than is used in most evolutionary search procedures. For each connection, its absence or presence required 1 bit, the encoding of the weight required 7 bits. The 7 bits were interpreted as a base-2 number; the value was mapped to a number between [-1, +1]. In the maximal network, there were a total of 1114 possible connections: (240+1) * 4 (input to hidden), and (4 +1) * 30 (hidden to output). The (+1) factors are used for connections to a *bias* input unit, this is a unit whose value is permanently clamped to a value of (+1.0), to which all units are connected [12]. With 1114 connections, the total size of the bit string was 8912 bits.

In order to give a base-line performance from which to compare the performance of the evolutionary algorithms, results using the backpropagation algorithm are also given. The backpropagation algorithm incorporated all of the modifications which are used in standard ALVINN training [15]. All of the results presented in this chapter are the average of at least 10 training sessions. Backpropagation 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%. The final networks successfully pruned away, on average, approximately half of the possible connections. The drawback of the PBIL approach, however, 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. As mentioned before, the need for generalizability must be carefully weighed with the desire for efficiency when choosing the appropriate training method. In Figure 1.8, sample input and output are given after training by PBIL. Samples attained by backpropagation are 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. 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. Further, the connections can be effectively 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 network was encoded in 7798 bits. This is smaller than the encoding used in the previous experiment as each connections was assumed to be present.

PBIL Backpropagation

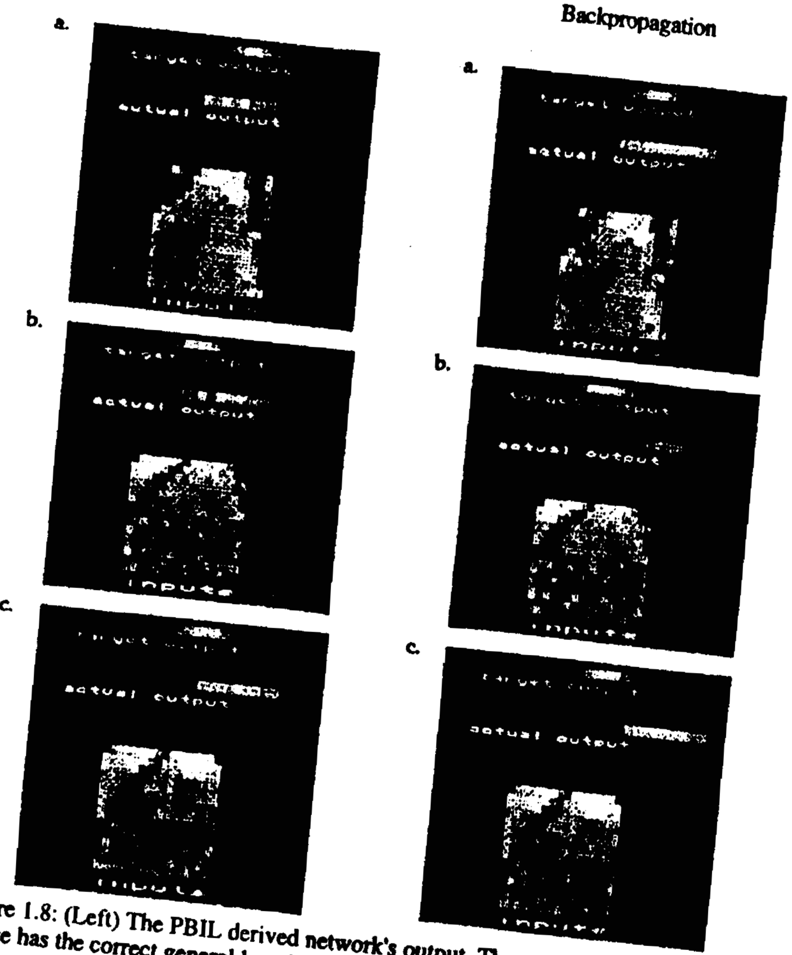

Figure 1.8: (Left) The PBIL derived network's output. The actual output for this image has the correct general location as the target output. However, the output is not as smooth as the target. (Right) Typical outputs of a network architecture trained with backpropagation. Figures are taken from the test set. Sum squared errors are as follows (PBIL) a: 4.2, b: 8.2, c: 9.9. and (Backpropagation) a: 2.0, b: 15.0, c: 23.7. Images chosen from the test set to show a wide range of errors.

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 absolute 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 is a measure of the distance (in output units) between the correct peak in the Gaussian, and the predicted peak in the Gaussian. This is a linear translation of the steering error. The performances of the training methods gauged by this error metric are also provided in Table 1.1.

30 Output	TRAINING METHOD		
	PBIL topology & weights	PBIL weights	Back-Propagation
SSE	8.19	7.96	9.40
% ERROR DECREASE 30 output BP (SSE)	12.9	15.3	n/a
GAUSSIAN PEAK POSITION ERROR (GPPE)	2.96	2.90	3.35
% ERROR DECREASE 30 output BP (GPPE)	11.6	13.4	n/a

Table 1.1: Results with training using the SSE error metric.

The errors are shown in Table 1.1 in terms of both average sum squared error and Gaussian peak position error. Since the evolutionary technique only used network evaluations based upon sum squared error (SSE), larger improvements are seen when performance is gauged using this error metric than when gauged with the GPPE error metric. Nonetheless, the GPPE error metric is more indicative of the actual performance of the system. In the next section, network evaluations which are based upon the GPPE metric are explored. It should be emphasized that although simple backpropagation could not have easily used the GPPE error metric, evolutionary algorithms can very easily incorporate such information to guide the search.

1.6.1 Using a Task Specific Error Metric

To this point, all of the experiments have used the sum squared error metric to guide the evolutionary search. Nonetheless, for many problems, the output which is produced by the 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, the peak difference error measure 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 [11]. 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 CFM error metric focuses effort towards performing the underlying task of classification rather than reproducing the exact target vector. Similarly, using the GPPE error metric focuses the training towards 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 reduce the GPPE error on each image in the training set, without regard to the sum squared error.

Using 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 determining where the peak of the Gaussian lies in the output vector, many of the small amounts of noise can be 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 Figure 1.9.

The average error using the GPPE error metric was 2.76 (GPPE); this is approximately an 18% improvement over backpropagation. The error measured in terms of SSE, however, was much higher than in the previous experiments: 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.

1.7 Conclusions

Various parameter settings were used for all of the training methods. In addition, the backpropagation algorithm used in this study maintained spatial information about the input retina, through neighbor weight smoothing, which the evolutionary algorithm was not given. Nonetheless, the evolutionary approach performed better, on average, than backpropagation. Although the evolutionary approach provides performance improvements, it also incurs severe computation penalties. For example, the backpropagation method was able to achieve its minimum in several minutes. The evolutionary approach took over an hour. In deciding whether to use an evolutionary approach or backpropagation, it is necessary to carefully weigh the need for accuracy with the desire for speed.

This chapter has presented several techniques for increasing the efficiency of evolutionary procedures for training artificial neural networks. The first is the evolutionary procedure used — the population based learning algorithm. Although it is far less complex than a simple genetic algorithm, it is effective in the optimization problems to which genetic algorithms are often applied [4]. The second is the evaluation of each network on a subset of the original training set.

The majority of the time in the evolutionary search procedure is spent evaluating the effectiveness of each network. The time to create a new potential solution vector, including the generation of a new solution vector, and the translation of it to the structure and weights of the ANN, is very small in comparison. Therefore, a reduction in the time spent in the network evaluation portion of the algorithm has a tremendous impact on the overall speed of learning.

PBIL using Gaussian Peak Position Error

Backpropagation using Sum Squared Errors

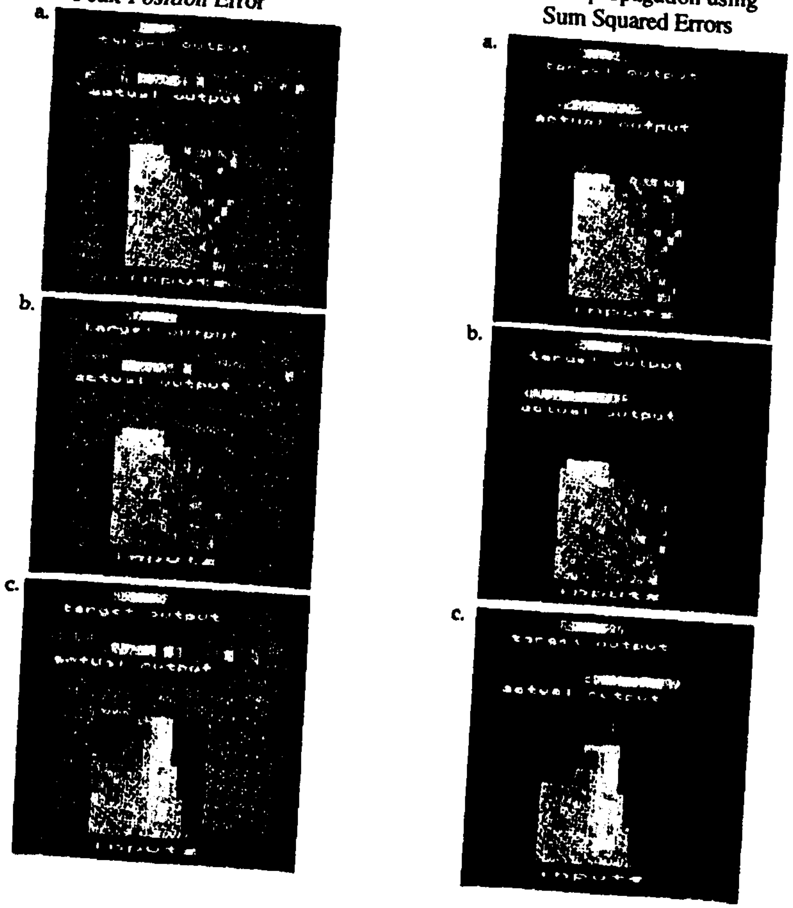

Figure 1.9: Sample input and output using the GPPE error metric. Images taken from test set. Images were chosen to show different amounts of GPPE errors: image A: 0.53, B: 1.16, C: 1.71. Backpropagation, trained with the standard SSE error metric, achieves the following GPPE errors: 0.49, B: 1.70, C: 5.12.

Evolutionary search procedures have the ability to use direct information of error metrics other than those which can easily be used by backpropagation. This can lead to improved performance on the specific task. In this chapter, either the

GPPE or SSE error metric was used to guide the evolutionary search. Although it was not tried in this chapter, perhaps using a combination of both the error metrics may give more information from which to guide the search, and therefore to find better solutions. This direction is open for future research.

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 chapter, which is similar to the one chosen by [14], is a hybrid between these two extremes. In this method, a maximal network is specified. The evolutionary algorithm searches for the network topology by either maintaining or throwing away possible connections. The network topologies which are evolved in this study are efficient; they use approximately only half of the total number of possible connections. This hybrid approach allows any *a priori* information regarding the network topology to be easily incorporated into the search. Although a more automatic specification of the network would be desirable, it is suspected that as the need for more automatic specification of networks is increased, using genetic or PBIL search methods will increase the already large search times.

1.8 Future Directions

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 2, 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 [8].

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 samples from the training set has the potential to greatly reduce the amount of time used for developing networks with good generalization capabilities, more exploration should be done to determine how much of the training set should be used for individual evaluations.

A promising area for future research is the integration of backpropagation with evolutionary search procedures. Evolutionary search has the ability to perform global search while backpropagation provides the ability to perform local optimizations. Therefore, backpropagation could be used as a "post processing" step after the EA is completed. Another benefit for the integration of these two procedures is potential reduction in the time needed for search by the EA. For example, backpropagation can be periodically used to locally optimize the best networks found through search. This makes the goal of the evolutionary search to find a good basin of attraction, and let backpropagation optimize the neural network once a good basin is found. Such an approach is taken by [13].

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, one of which is that of training networks with more than a single road type. The desire to use a small number of "road specific" 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 use the same inputs as the ALVINN networks, and 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.

Acknowledgements
I am grateful to Dean Pomerleau for his extended discussions of the ideas contained within this report. Thanks are also due to Charles Thorpe for his discussions and encouragement. Todd Jochem kindly provided the sample images from which all training and testing were 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.

The author is supported by a National Science Foundation Graduate Fellowship. 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, ARPA, or the U.S. Government.

References

[1] Abu-Mostafa, Y., Information Theory, Complexity and Neural Networks, *IEEE Communications Magazine.* Vol. 27, No. 11, 1989.

[2] Angeline, P., Saunders, G. & Pollack, J., An Evolutionary Algorithm that Constructs Recurrent Neural Networks. *IEEE Transaction on Neural Networks.* Vol. 5, No. 1, January, 1994.

[3] Baeck, T. & Schwefel, H.P. (1994) An Overview of Evolutionary Algorithms for Parameter Optimization, in *Evolutionary Computation.* Vol. 1, No. 1, pp. 1-24, 1993.

[4] Baluja, S. (1994) Population Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning. Carnegie-Mellon University Technical Report, CMU-CS-94-163.

[5] Baluja, S. (1994) Evolution of an Artificial Neural Network Based Autonomous Vehicle Controller. Paper in Progress.

[6] Belew, R. (1993) Interposing an Ontogenic Model Between Genetic Algorithms and Neural Networks. In Hanson, Cowan, Giles (ed). *Advances in Neural Information Processing Systems* 5. Morgan Kaufmann Publishers, San Mateo, CA 99-106.

[7] Fogel, D.B. (1994) An Introduction to Simulated Evolutionary Optimization. *IEEE Transaction on Neural Networks.* Vol. 5, No. 1, January, 1994.

[8] Fogel, D.B., Fogel, L.J. & Porto, W. (1990) Evolving Neural Networks. *Biological Cybernetics* 63. 487-493.

[9] Goldberg, D. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning.* Reading, MA: Addison-Wesley Publishing Company.

[10] Gruau, F. (1993) "Genetic Synthesis of Modular Neural Networks". In Forrest, S. (ed). *Proceedings of the Fifth International Conference on Genetic Algorithms.* 318-325. Morgan Kaufmann Publishers, San Mateo, CA.

[11] Hampshire, J.B. & Waibel, A.H., (1989) "A Novel Objective Function for Improved Phoneme Recognition Using Time-Delay Neural Networks". Carnegie Mellon University Technical Report. CMU-CS-89-118.

[12] Hertz, J., Krogh, A., & Palmer, R.G. (1993) *Introduction to the Theory of Neural Computation,* Addison-Wesley, Reading, MA.

[13] Keesing, R. & Stork, D. (1991) "Evolution and Learning in Neural Networks: The Number and Distribution of Learning Trials Affect the Rate of Evolution". In Lippman, Moody, Touretzky (ed). *Advances in Neural Information Processing Systems 3.* Morgan Kaufmann Publishers, San Mateo, CA.

[14] Maniezzo, V. (1994) Genetic Evolution of the Topology and Weight Distribution of Neural Networks. *IEEE Transaction on Neural Networks.* Vol. 5, No. 1, January, 1994.

[15] Pomerleau, D.A. (1994) Personal Communication. Carnegie-Mellon University.

[16] Pomerleau, D.A. (1993) *Neural Network Perception for Mobile Robot Guidance,* Kluwer Academic Publishing.

[17] Pomerleau, D.A. (1992) Progress in Neural Network-based Vision for Autonomous Robot Driving. In the *Proceedings of the 1992 Intelligent Vehicles Conference,* I. Masaki (ed.), pp. 391-396.

[18] Pomerleau, D.A. (1991) Efficient Training of Artificial Neural Networks for Autonomous Navigation. In *Neural Computation* 3:1 pp. 88-97.