Applying Advanced Learning Algorithms to ALVINN

Parag H. Batavia
Dean A. Pomerleau
Charles E. Thorpe

9 October 1996
CMU-RI-TR-96-31

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-3890

Abstract

ALVINN (Autonomous Land Vehicle in a Neural Net) is a Backpropagation trained neural network which is capable of autonomously steering a vehicle in road and highway environments. Although ALVINN is fairly robust, one of the problems with it has been the time it takes to train. As the vehicle is capable of on-line learning, the driver has to drive the car for about 2 minutes before the network is capable of autonomous operation. One reason for this is the use of Backprop. In this report, we describe the original ALVINN system, and then look at three alternative training methods - Quickprop, Cascade Correlation, and Cascade 2. We then run a series of trials using Quickprop, Cascade Correlation and Cascade2, and compare them to a BackProp baseline. Finally, a hidden unit analysis is performed to determine what the network is learning.

This research was partially supported under a National Science Foundation Graduate Fellowship, the National Highway Traffic Safety Administration (NHTSA) under contract no. DTNH22-93-C-07023, and by USDOT under Cooperative Agreement Number DTFH61-94-X-00001 as part of the National Automated Highway System Consortium

Keywords: ALVINN, Autonomous Land Vehicle, Backpropagation, Quickprop, Cascade Correlation, Cascor, Cascade2.

1. Introduction

ALVINN [1] (Autonomous Land Vehicle in a Neural Net) is a backprop based neural architecture which has been used to successfully drive a car along a highway at highway speeds. The input to ALVINN is a 30x32 image taken by a camera connected to a digitizer. The output is a steering direction, which is accomplished by a motor connected to the steering column. One of the main concerns with ALVINN has been the speed at which it learns to drive in new situations. While it is capable of on-line learning, the training time is usually on the order of 2-3 minutes. The speed is influenced by many factors: the training algorithm, the training set size, the dynamic training set management, and the computational power available. In the 5 years since the original ALVINN work was performed, not only has computational power increased, other learning methods have also been developed which may allow a speedup in learning.

The motivation for looking at this is to see whether it is viable to return to actively using ALVINN as a supplement to the RALPH (Rapid Adaptive Lateral Position Handler) [2] lane tracking system on Navlab 5. One of the main reasons ALVINN was replaced was the long learning time when dealing with new road conditions, coupled with the requirement of driver intervention during that period. It may be possible to remove both constraints by using faster learning algorithms, and by using RALPH to train ALVINN when road conditions change. After RALPH trained ALVINN, the neural net’s generalization capabilities would ensure that we don’t have simply two copies of RALPH running.

To this end, we measure the performance of three learning algorithms on learning ALVINN. The algorithms are quickprop, cascade correlation, and cascade2. The performance of these three algorithms is compared against a backprop baseline. In addition, we also use hidden unit analysis to determine how the network learns.

2. Previous Work

There has not been a lot of work in improving ALVINN’s training time, although there has been work on improving the robustness and accuracy of ALVINN. Radial Basis Functions (RBF’s) have been applied to the ALVINN problem [3] with success, but no mention of training time was given. ELVIS (Eigenvectors for Land Vehicle Image System) [4] attempted to discover *why* ALVINN works. It learned the eigenvectors of an input image and corresponding steering direction using principle components analysis. New images were projected into this eigenspace to generate a steering output. ELVIS worked nearly as well as ALVINN, demonstrating that a neural network wasn’t necessary for this particular task. The MANIAC (Multiple ALVINN Networks in Autonomous Control) [5] system improved the robustness of ALVINN by using an architecture which combined the hidden unit outputs of multiple ALVINN networks trained for different road types. Generalization improved, as the system was able to drive on a four lane road when it had only been trained on 1 and 2 lane roads.

3. ALVINN

ALVINN is trained with a standard backprop algorithm, with certain modifications. A momentum term is included, to escape small local minima. The flat spot problem, in which the derivative of the sigmoid goes to zero at the boundaries, is solved by adding a constant term of 0.1 to each calculation. Because of the large number of input units compared to hidden and output units, each layer's initial random weights and learning rate is divided by the fan-in of the layer that it projects to. Weight decay and a learning/momentum rate schedule are also used.

Rather than having a single linear output indicating steering direction, the output is a 20 unit vector which models a gaussian whose peak is the steering direction. In general, networks such as these are easier to train than one in which there is a single linear output. A gaussian is used to allow for "sub-neuron" accuracy in the steering direction. This means that the peak of the gaussian is allowed to fall in between two output neurons - the peak of the gaussian is computed by taking a weighted centroid centered around the neuron with highest activation.

In the on-line version of ALVINN, sophisticated training set management was done to avoid biasing the network towards recent experience. Another problem that needed to be solved was ensuring that the network would not learn a bias towards steering in one direction. Two techniques were used for this: image buffering and image transformation.

Image buffering involved maintaining a buffer of previously viewed images. This was to ensure that as new situations are encountered, older experience is not forgotten. When a new training pattern is captured, an older one must be replaced. The system removes a pattern for which the network is already performing well. This led to the problem that if the human driver steered incorrectly when a pattern was grabbed, it would most likely remain in the buffer indefinitely. This was solved by adding a random replacement probability to every pattern in the buffer. To prevent biasing the network towards left or right terms, a further constraint was added to ensure that the mean steering direction of all the patterns in the buffer is straight ahead.

Image transformation addressed the problem that when a human drives, he rarely encounters very unusual situations. Most likely, the driver may be a bit to the left or right of lane center, but it is rare for him to be at more than a 5 or 6 degree angle relative to the lane. The network, however, has to be able to deal with situations like this. Therefore, a geometric transform is applied to each captured pattern, adding rotation and translation. A pure pursuit model [10] is used to generate the correct steering direction for these artificial patterns. If the required turn is outside the range the network can represent, it is not inserted into the pattern buffer.

The original backprop that trained ALVINN (which we use as a baseline in our comparisons) used a training buffer of 200 images, and a cross validation buffer of 50 images. Backprop would minimize the cross validation set error in about 50-75 epochs. For the purposes of these experiments, all training is off line. We also used a 200 element artificially generated balanced training set, along with a similarly balanced 50 element cross validation set. After training, a final evaluation is done with an independent 50 element test set. A complete explanation is given in [1].

4. Learning Algorithms

Four different learning algorithms were chosen for investigation: backprop (as a baseline), quickprop, Cascade Correlation, and Cascade II. The next four sections will introduce the algorithms and explain the basic ideas behind their implementations. For a thorough explanation of each method, see [1][6][7][8].

4.1. Backpropagation

The backpropagation algorithm is the standard baseline learning algorithm used with fully-connected, layered, feedforward networks. The activation function on each layer is TanH, since ALVINN's outputs range from (-1, 1). For these experiments, the network will have 625 input units (corresponding to a 25x25 input image), 4 hidden units (except in the case of cascor, in which the hidden units are added as needed), and 20 output units. Training is done in batch mode, using momentum and flat spot elimination. To improve the learning rate, the initial weights and learning rate of each layer is divided by the fan-in of the layer it projects to. Another heuristic used is weight decay, which prevents weights from growing unbounded by reducing the magnitude of a weight by a small percentage of its previous value. This is important particularly because of the large number of weights involved. A training schedule is also used to gradually ramp up the learning rate and momentum to a preset maximum.

4.2. Quickprop

Standard BackProp is a gradient descent algorithm which takes steps proportional to a learning rate parameter. This is not always optimal, as it may be possible to take much larger steps, which would improve overall learning speed. Methods like momentum and learning rate schedules are some heuristics people have tried to achieve this effect. Scott Fahlman's Quickprop [7] takes a different approach. Quickprop makes two assumptions about the error surface: 1) the error vs. weight curve can be approximated by a parabola whose arms open upwards, and 2) the required change in slope in one dimension is not affected by the simultaneous changes in other dimensions in weight space. Using these two assumptions, the weight vector which leads to the global minimum in this parabolic space is:

$$\Delta w(t) = \frac{S(t)}{S(t-1)-S(t)} \times \Delta w(t-1)$$

Where S(t) and S(t-1) are the values of $\frac{\partial E}{\partial w}$ at times *t* and *t-1*. Even though this is a crude approximation, it is very fast to compute, and empirically works well. In certain cases, the weight update can tend towards infinity. This can happen when the current slope is in the same direction as the previous slope, but larger in magnitude. To handle this, a parameter μ is introduced, which acts as a "maximum growth factor". The weights cannot change by more than μ times the previous weight change. In most cases, μ is on the order of 1.75. Another problem occurs when the slope in a direction is close to zero - in this case, the weight tends not to change, even if new conditions exist which would require it to change. This is fixed by adding another parameter, ε, which is multiplied times the current slope and added to the weight update value computed by the quadratic formula. Neither of the above two parameters are very sensitive. For most problems, there is a fairly large range of values which work well.

4.3. Cascade-Correlation

Cascade-Correlation (Cascor) is another algorithm developed by Scott Fahlman [6]. Cascor was developed to address two problems with backprop: step size, and the moving target problem. The step size problem is the same as discussed in Sec. 3.2. The moving target problem is that each hidden unit in a feed-forward network is trying to independently develop into a feature detector, while weights are changing around them, and without any cooperation or coordination. This can lead to something called the *herd-effect*. This is when a set of hidden units concentrate on one problem, to the exclusion of another. All the units concentrate on solving that problem, and when they succeed, they realize that the major source of remaining error is another problem. The hidden units then all migrate towards solving that problem, abandoning the first. This leads to all the units oscillating between detecting different features.

Cascor solves this problem by only allowing one hidden unit to evolve at a time, and forcing each layer to have only one hidden unit. Each hidden unit is fully connected to the input layer, and also all the hidden layers before it.

The network starts with no hidden units, and no connections between the input and output layers. This is not how cascor is normally run. However, given that we have 626 inputs (625 + 1 bias unit), and 20 outputs, there would be 12,520 connections that would have to be calculated each epoch. For the sake of speed, these connections were eliminated.

This network is then trained using Quickprop (which acts as a faster version of LMS when there are no hidden layers). At some point, training will reach a plateau. If there is no significant improvement over a certain number of epochs (specified by a patience factor), the algorithm adds a hidden unit.

When training to add a hidden unit, a set of hidden units is created, with each “candidate” unit fully connected on the input side, but not connected at all on the output side. The input side is fully trainable. The candidates are trained to maximize *S*, which is the sum over all output units of the magnitude of the correlation between the candidate units value and the residual error of the net. The partial of *S* with respect to the candidate units incoming weights can be derived similarly to the backprop derivation. Quickprop is used to perform an *ascent* of *S*, as we want to maximize the correlation. Again, a patience parameter is used to determine when candidate training is done. The unit out of the set of candidates with the highest correlation score is added to the net. If the correlation is positive, the initial output weights are negative, which would cancel out some of the residual error. Similarly, if the correlation is negative, the initial output weights are positive. Finally, the outputs of the net are trained again, until either the error becomes sufficiently low, or another plateau is reached, in which case another hidden layer will be added.

There is still the remaining problem of setting an initial condition for the first correlation cycle. If there are no initial input to output connections to train, there will be no error for the candidate training phase to correlate with. This was solved by setting the initial network outputs to all 0s. Therefore, the error of the network will always be the negative of the desired output. The candidate phase then correlates to this error, and negates the output weights when installing the first hidden unit.

4.4. Cascade 2

Cascade 2 is a new algorithm, also developed by Scott Fahlman [Scott Fahlman, personal communication]. The output training phase is the same as in cascor. However, candidate training is changed. In cascor, the correlation between the candidate units’ output and all the network output units is maximized. In cascade 2, the correlation is removed, and trainable output weights are added to the candidate. The unit is trained to minimize the difference between candidate output sum squared error and network residual sum squared error. These output weights are then copied into the original network. This means that two layers need to be trained for each candidate unit. The advantage, however, is a new candidate which better reduces the residual network error than in cascor. This leads to fewer output epochs.

5. Experiments

To find out how well the above algorithms work in training ALVINN, we ran a series of 50 trials of each method. The training set was 200 artificially generated 1 and 2 lane road images. The mean steering direction of all the images in the training set was straight ahead. The cross validation and test set were 50 patterns each, drawn from the same distribution as the training set.

The trials were run by modifying publicly available code for backprop, quickprop, cascor, and cascade2. Except for cascor and cascade2, each trial was run until the cross validation error either plateaued, or increased. For cascor, however, we did not run a cross validation at each epoch. This is because cascor looks for a plateau in the training set error to determine when a new unit should be added. Therefore, there are three termination conditions for cascor: 1) the error of the training set drops below a certain threshold, 2) a cross validation is performed after every *cycle* (i.e., one full output training and candidate selection generation), and stops after adding hidden units no longer affects the error, or 3) the maximum number of allowable hidden units have been added. Therefore, we had to pick a training set error threshold below which training would be considered done. This was done by looking at the average training set error of the completed quickprop trial, and using that value as a beginning for the termination condition. In addition, both cascor and cascade2 were limited to 4 new hidden units. Trials were run with fewer (3) and more (up to 6) hidden units, but there was no improvement in performance with 5 or 6 hidden units, and a decrease in performance with only 3.

Rather than using the mean squared error of the test sets, we used the mean absolute peak difference error. This is because we are concerned with steering direction, rather than actual output values. Therefore, the peak of the output gaussian is important. The peak is found by taking a weighted centroid centered on the unit with highest activation. Rather than looking at the entire output vector, we look at only a window. This is because in the case of a fork in the road, the network output can have two peaks. The centroid of two peaks would be somewhere in the valley between them. The window size we used was 4 units on each side of the center. This peak was then compared to precomputed peaks for the test set.

Each algorithm has a set of parameters that needs to be adjusted. This part of training is considered to be something of a black art, as the best values for the different parameters depend greatly on what is being learned. We picked these parameters by running many trials for each algorithm, adjusting one parameter at a time, until we were able find (what we hope is) the best performance. After this, we ran the 50 trials. The goal is to compare each algorithm at its best, to avoid biasing the experiments. For backprop, the learning rate, momentum, decay rate, and ramping rates have to be set. For these experiments we used a learning rate of 0.015, a momentum term of 0.9, and we ramped up the learning rate and momentum using a rate term of 0.05. This means that the learning rate and momentum increase linearly over 20 epochs until they reach their maximum value (0.015 and 0.9, respectively). We also used a weight decay term of 0.0001, which means that after each iteration, all weights were reduced by 0.01%. A constant term of 0.1 was added to each sigmoid derivative to avoid the flat spot problem.

For quickprop, we used a learning rate of 0.015, a maximum growth parameter of 1.75, and a weight decay term of 0.0001. For cascor, the output epsilon was 4.0, the maximum growth parameter was 1.75, and weight decay was 0.0001. The candidate epsilon was 10.0, and its maximum growth parameter was 1.75. Additionally, we set a limit of 4 hidden layers, and stopped output training when the peak difference over the previous 2 cycles was less than 1%. The output units had a patience value of 8 epochs, and the candidate units were set at 4 epochs.

Cascade2 has a few more parameters to set. For output training, epsilon was 6.0, max. growth was 1.75, and patience was 8. The candidates have two sets of parameters - one for the input side and one for the output side. The input side used an epsilon of 1.0 and a max. growth of 1.75. The output side used the same numbers. Patience was set to be the same as in cascor. In all cases, trials were run on a 133 Mhz Pentium processor under the Linux operating system.

6. Results

The results are shown in table 1:

	Avg. Epochs	Epochs/ Second	Seconds/ Trial	Max. Test Peak	Min. Test Peak	Avg. Test Peak	Peak Std. Dev.
BackProp	32	3.81	8.42	0.7000	0.3800	0.5488	0.0622
QuickProp	18	3.77	5.04	0.6909	0.3695	0.5087	0.0748
Cascor	163	4.27	38.30	0.9800	0.5400	0.6964	0.0927
Cascade 2	152	5.85	25.96	1.3315	0.8086	1.0534	0.1168

Table 1: Final Results over 50 Trials

Of the three algorithms, quickprop was both the fastest and had the best performance, although not significantly so. Backprop was second in time and performance, with cascade2 3rd in time but last in performance. Cascor was the opposite, being 3rd in performance, but fourth in time.

These results were a surprise. Fahlman [7] notes up to a 10x speedup (in epochs) between quickprop and backprop for a 10-5-10 complemented encoder. However, the backprop we used was considerably "tweaked" to provide the best performance. As mentioned in Section 4.1., the use of momentum, sigmoid prime offset, and a learning schedule did much to improve the learning speed of backprop. Using standard vanilla backprop, it is easy to believe that quickprop would be 10 times faster.

What is highly unusual is the performance of cascor and cascade2. In both the non-constructive algorithms, training time was very fast, as measured in epochs. Constructive algorithms such as cascor and cascade2 work in cycles, alternating between training outputs and training candidates. For performance, the maximum number of epochs per cycle was set to 20 for the outputs, and 40 for the candidates. A large enough patience had to be used so as not to stop training once a small local minimum was found. We did try using a small patience, but that degraded performance significantly, as the hidden units never managed to "grab on" and their correlation and fitness scores were therefore low. This use of larger patience values requires a *minimum* of 12 epochs per cycle (8 for the output, and 4 for the candidates). Over four cycles, that's 48 epochs. This means that the *best* that could be done by cascor and cascade 2 would be to perform slightly worse than backprop (in epochs only - both cascade and cascor had higher epochs/second rates, as the number of connections start out low, and goes up as hidden units are added). It has been suggested [Scott Fahlman, personal communication] that these two algorithms may perform better given a single unit output representation. Therefore, rather than having a 20 unit gaussian output, the network would have a 1 unit linear output, indicating steering direction. It has been shown in [1] that a single unit output does not perform as well as a distributed output.

What is interesting to note is the obvious effect that the past few years have had on computation speed. The original ALVINN took anywhere from 2 to 3 minutes to train, under similar circumstances. It had a larger input retina (30x32), which is about a 60% increase in the number of connections. Adding 60% to the time for backprop results in a training time of about 13 seconds. Also, by using off-line training we are avoiding the training set management that has to be done. Even so, the times are still much lower than could be explained by that alone. To demonstrate this, we ran one trial of backprop on a Sparc IPX, which is similar to the Sparc which was originally used. One run took 81 seconds.

Another reason for a training time of 2-3 minutes is that enough novel conditions have to be encountered while driving for the network to generalize well. This can take a couple of minutes. However, during that time, quickprop would allow more epochs of training than backprop, although due to quickprop's nature, there would not be any benefit to do so.

7. Hidden Unit Analysis

Now that we have seen how the different networks have performed, it is interested to examine why they perform as they do. One common technique used is weight analysis. This is an especially useful technique where the inputs have spatial organization, as road images do. Figures 1-30 show weight diagrams for all four algorithms, and test images for the three new ones. The weight diagrams show the four sets of input->hidden layer weights, organized in the same 25x25 pixel array as the input images are. This shows (by holding the paper about a foot away and squinting) what features each hidden unit is sensitive to. The hidden->output weights are also shown for each hidden unit. This tells the direction that hidden unit will vote for when it is activated. As both one and two lane roads are used in the training set, features should be present which are generic enough to handle both types of roads. A hidden unit analysis of the backprop network is already given in [1], and these results are similar to the ones described below. The original weight diagrams are given in Figures 1-4. Structure is visible, but hard to detect.

Therefore, the weight diagrams are modified slightly. A histogram of the weights is calculated, and the intensity of the top 10% of values are all converted to white. The bottom 10% are converted to black. The remaining 80% are stretched linearly over the remaining 254 gray levels. Figures 4-8 and 15-18 show the result of this. In these diagrams, structure is more apparent. For instance, in figure 6, which shows hidden unit 2, two areas of activation are apparent. There are also two peaks in the hidden->output weight vector of that hidden unit. These two peaks correspond well with the placement of the two areas of high activation in the input->hidden layer. One interpretation is that if the input road is shifted to the left, this hidden unit would vote for both a turn to the left and right, while voting against going straight ahead. If the input was a straight, centered road image, then the overall activation of this hidden unit would be to steer straight ahead. Similarly, hidden unit one (figure 5) prefers a right turn. If a right shifted road image was presented, the hidden unit activation would be high. However, if the road were left shifted, hidden unit one would have low activation. Again, these results qualitatively match those of the original ALVINN.

The quickprop weights (figures 9-12, 15-18) show something very different. There is very little structure visible, and enhancing the images did not bring out much hidden structure, except in unit 1. The weights all have very similar magnitudes, except for small areas of high activation. These areas also changed location between different runs, whereas the backprop areas of emphasis looked similar between runs. Histograms of both quickprop and backprop diagrams were computed, and are shown in figures 31-38. The quickprop histograms are narrower and higher than the backprop ones, and also have more outliers.

This lack of structure would seem to imply poor generalization of the net. However, as was shown in the results, quickprop actually performed slightly better than backprop, although this is not statistically significant. The question then is whether there is something intrinsically significant about the small areas of high activation in the quickprop weights. To test this, we looked at the correlation between two variables. The first is the activation (in each test image) of the pixel corresponding to the high magnitude weight in each hidden unit. The second is the distance between the peak steering direction of each training pattern and the peak steering direction of the hidden->output weight vector of the hidden unit being tested. If there is a high correlation[1] between the two events, then the network is picking up on something significant about that small area, and uses that to generate the steering direction, with minimal input from the rest of the image.

We ran a set of experiments, computing the correlation between the intensity of the pixel corresponding to the weight with the largest positive value, the largest negative value, and a random pixel, with the difference in peak steering direction, over all 200 training set patterns. The results for five trials are shown in table two:

1. We're actually looking for a negative correlation, as an increase in pixel magnitude should cause a decrease in peak steering distance.

	Hid. 1 Max Corr.	Hid. 1 Min Corr.	Hid. 1 Rand. Corr.	Hid. 2 Max Corr.	Hid. 2 Min Corr.	Hid. 2 Rand. Corr.	Hid. 3 Max Corr.	Hid. 3 Min Corr.	Hid. 3 Rand. Corr.	Hid. 4 Max Corr.	Hid. 4 Min Corr.	Hid. 4 Rand. Corr
Trial 1	0.01	-0.21	-0.09	-0.12	0.17	0.08	-0.03	0.17	0.03	-0.03	0.30	0.04
Trial 2	0.11	0.06	-0.17	-0.06	-0.10	-0.11	-0.26	-0.13	-0.16	-0.28	-0.10	0.44
Trial 3	0.16	0.02	-0.03	0.07	-0.15	0.08	-0.10	-0.07	0.06	-0.04	-0.01	-0.20
Trial 4	-.020	-0.18	0.33	0.18	-0.05	0.03	-0.26	-0.11	0.14	-0.07	0.00	-0.00
Trial 5	-0.03	-0.02	-0.00	-0.08	-0.11	-0.26	-0.14	0.13	-0.29	-0.26	0.03	-0.08

Table 2: Correlation Test Results

Table 2 shows the results of 5 trials. The correlation between the maximum, minimum, and random pixel vs. the peak steering direction are shown. The correlation values for all cases are low. In some cases, the random pixel has a higher correlation coefficient than either the maximal or minimal pixel. All the number are low enough however, that it is fair to say that there is no correlation in any of the three cases.

If there is no correlation, then why are there these outliers in the weight histogram? One possibility is the approximations quickprop makes when training. There is a lot of noise in the calculations, due to the parabolic approximation to the second derivative of the error surface. Perhaps, this allows individual weights to grow large. One problem with quickprop has been that weights can grow large enough to cause overflows. Normally, weight decay takes care of this, but it may still not be enough to prevent some weights from growing. However, even these larger weights are still no where near causing floating point overflow. Most likely, quickprop generalizes well because there is some very faint structure in the weights. The first hidden unit in quickprop (figure 9) shows some possibility of this. There are two light areas, one on either side of the image.

The above experiment determined that there is no correlation between and individual pixel, and the distance between the peak steering direction and the hidden units preferred steering direction. We also computed the SSD between each training set output and the hidden->output weight vector for each hidden unit. Then, we computed the correlation between the SSD for each pattern and the activation of the maximum activated, minimum activated, and a random pixel in the input->hidden weight vector. To look at the importance of the individual pixel, we also looked at the immediate top and left neighbors of the maximal and minimal pixel. This gives us a better idea of each hidden unit's contribution to the total output - i.e. a hidden unit *could* be activated by the maximum pixel, but its output could be "overruled" by another hidden unit with higher (or lower) activation. The results of this are given in tables 3-6.

	Max Corr.	Max Top Corr.	Max Left Corr.	Min Corr.	Min Top Corr.	Min Left Corr.	Rand. Corr.
Trial 1	-0.10	-0.26	-0.27	-0.15	-0.13	-0.10	0.24
Trial 2	-0.56	-0.51	-0.50	-0.24	-0.56	-0.07	0.21
Trial 3	-0.04	-0.03	0.04	-0.06	-0.08	0.20	0.11
Trial 4	-0.71	-0.69	-0.70	-0.20	-0.23	-0.03	-0.48
Trial 5	-0.42	-0.51	-0.45	-0.24	-0.25	-0.08	0.12

Table 3: SSD Correlation Test Results - Hidden Unit 1

	Max Corr.	Max Top Corr.	Max Left Corr.	Min Corr.	Min Top Corr.	Min Left Corr.	Rand. Corr.
Trial 1	0.28	0.23	0.20	0.10	0.09	0.27	-0.55
Trial 2	0.24	0.27	0.18	-0.20	-0.19	0.06	-0.05
Trial 3	0.59	0.63	0.53	0.02	0.02	0.15	0.16
Trial 4	-0.23	-0.27	-0.30	0.08	0.12	0.12	-0.01
Trial 5	-0.12	0.07	0.00	-0.05	-0.07	0.21	0.12

Table 4: SSD Correlation Test Results - Hidden Unit 2

	Max Corr.	Max Top Corr.	Max Left Corr.	Min Corr.	Min Top Corr.	Min Left Corr.	Rand. Corr.
Trial 1	-0.10	-0.09	-0.16	0.20	0.29	0.34	-0.60
Trial 2	-0.15	-0.12	-0.04	0.22	0.23	0.23	0.20
Trial 3	-0.02	-0.02	-0.16	-0.00	0.01	0.01	0.40
Trial 4	-0.04	-0.02	-0.25	-0.26	-0.26	-0.24	0.62
Trial 5	-0.02	-0.03	-0.21	0.75	0.70	0.71	0.05

Table 5: SSD Correlation Test Results - Hidden Unit 3

	Max Corr.	Max Top Corr.	Max Left Corr.	Min Corr.	Min Top Corr.	Min Left Corr.	Rand. Corr.
Trial 1	0.58	0.61	0.52	0.68	0.66	0.66	-0.39
Trial 2	-0.09	-0.07	-0.26	-0.17	-0.13	-0.22	-0.36
Trial 3	-0.09	-0.07	-0.26	-0.14	-0.12	-0.21	-0.03
Trial 4	-0.03	-0.03	-0.26	0.56	0.57	0.58	-0.36
Trial 5	-0.71	-0.68	-0.71	-0.37	-0.49	-0.24	0.14

Table 6: SSD Correlation Test Results - Hidden Unit 4

What we would expect is a high negative correlation in the max pixels, a high positive correlation in the min pixels, and a low absolute correlation for the random pixel. The results verify this in some cases. This indicates that the individual pixel *is* correlated with the output of the net. However, it is not the actual pixel itself that matters, but rather, the area that the pixel is in which is important. To demonstrate this, we looked at the activation correlation of the pixel to the left of and immediately above the maximally and minimally activated pixel (again, when talking about maximal and minimal pixels, we mean which *weight* in the input->hidden layer was appropriately activated). The correlations of these neighboring pixels is similar to the center pixels. This shows that what is most likely happening is that quickprop *is* finding some structure, but that it is very localized. What is interesting is the high correlation of the randomly selected pixel in some of the cases (trails 1 and 4 in Table 5, for instance). The random pixel was selected to be at least 8 pixels away from either the maximal or minimal pixel. Overall, the four tables of numbers do not paint a consistent picture. This further reinforces the conclusion that the individual pixels do not really matter. It is the area that they are in that is important

The other two learning algorithms, Cascor and Cascade2 both use quickprop as their learning algorithms. As expected, their weight diagrams look similar to those of quickprop. The first hidden unit in Cascade2, however, often settles to something similar to figure 25. The structure of a 2-lane road is evident, with the desired output being almost straight ahead. The rest of the units, however, show a similar lack of structure. This lack of structure is not indicative of poor performance, however. Because Cascor and Cascade2 both have cascaded inputs, the training tends to settle on the most significant feature of the input for the first hidden unit (as is demonstrated in figure 25). Following units could possibly be learning exceptions to that feature, and so forth. Therefore, the 2nd - 4th hidden units may not show any evident structure.

What can be seen, however, is the effect each hidden unit has on the network output after it has been added, and the output weights trained. Figures 23, 24, 29 and 30 show this. For Cascor and Cascade2, the output for the two lane road image is spread across the steering directions, and doesn't begin to come into "focus" until after the 3rd unit is added.

8. Conclusion

As in other tests [7][9] quickprop has shown itself to be a faster learning algorithm than backprop. This is reinforced by the fact that learning ALVINN is not a trivial challenge. Cascor and cascade2 did not do as well. It seems that the overhead in a constructive algorithm is not offset by better hidden unit representation for this problem. Most likely, the problems of backprop which Cascor were meant to address, mainly the step size and moving target effects, do not really affect ALVINN.

Although RALPH does a very good job of lane-keeping, the above experiments show that it may be feasible to augment RALPH with a newer, faster version of ALVINN. An extended Kalman filter [11] or other sensor fusion technique could be used to merge the results of the two processes, as both RALPH and ALVINN can provide the confidence measures required by sensor fusion algorithms.

Acknowledgments

The authors would like to thank Scott Fahlman for discussions which aided in the completion of this work.

9. Images

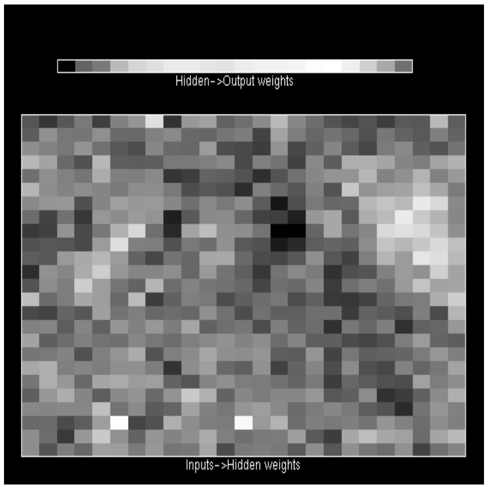

Figure 1: Hidden unit 1 weights

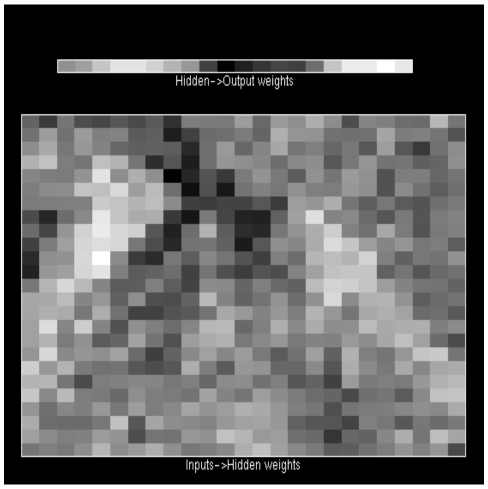

Figure 2: Hidden unit 2 weights

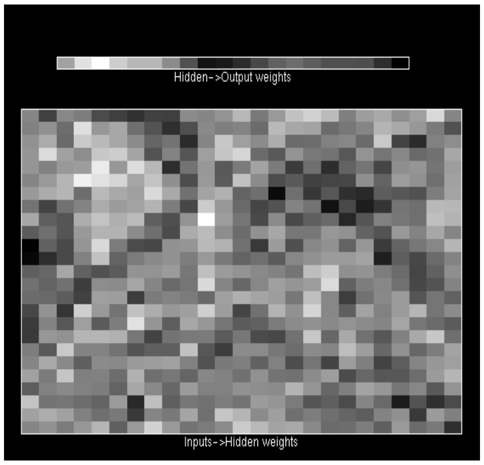

Figure 3: Hidden unit 3 weights

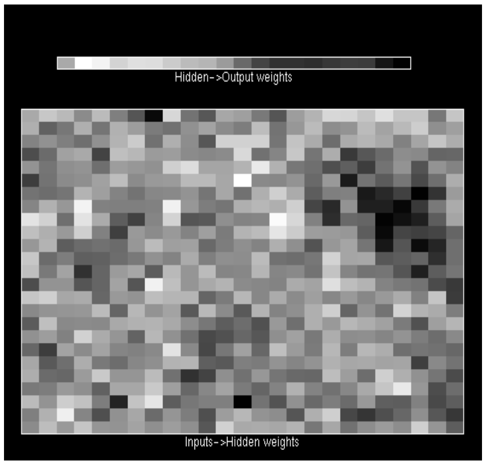

Figure 4: Hidden unit 4 weights

BackProp weight diagrams

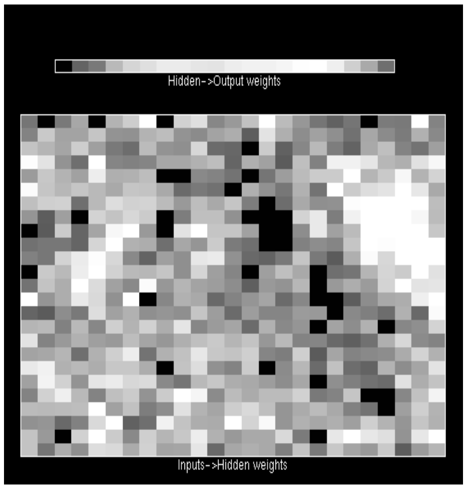

Figure 5: Hidden unit 1 weights

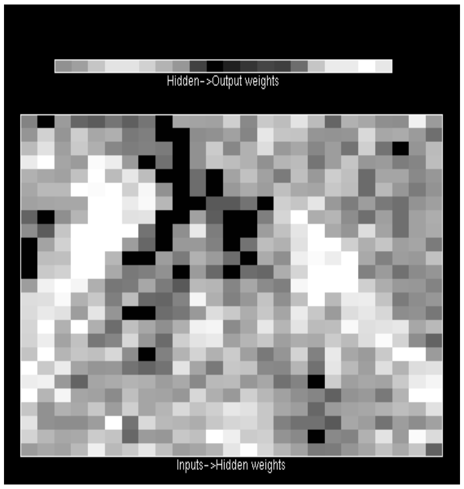

Figure 6: Hidden unit 2 weights

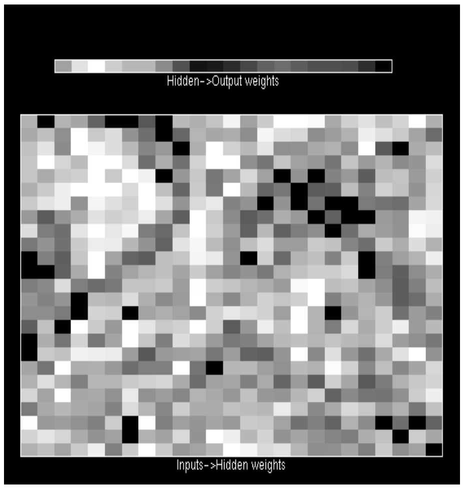

Figure 7: Hidden unit 3 weights

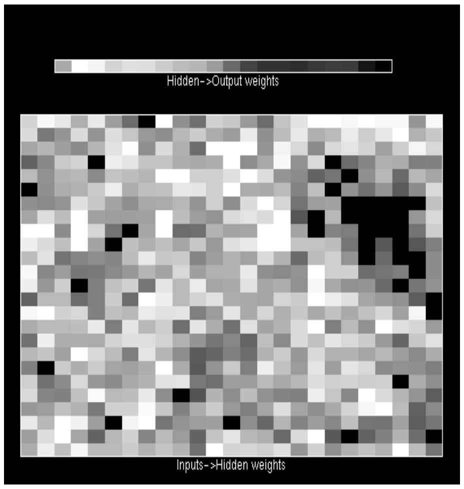

Figure 8: Hidden unit 4 weights

Enhanced Backprop weight diagrams

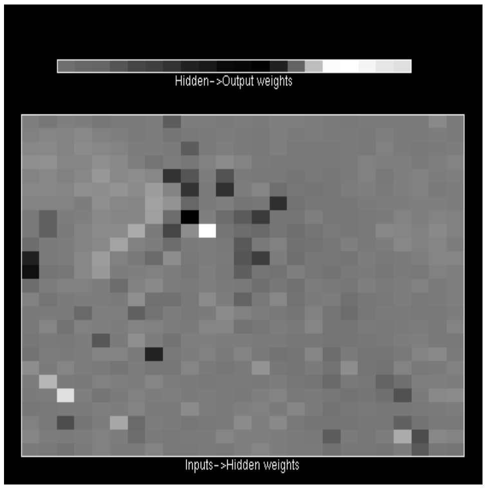

Figure 9: Hidden unit 1 weights

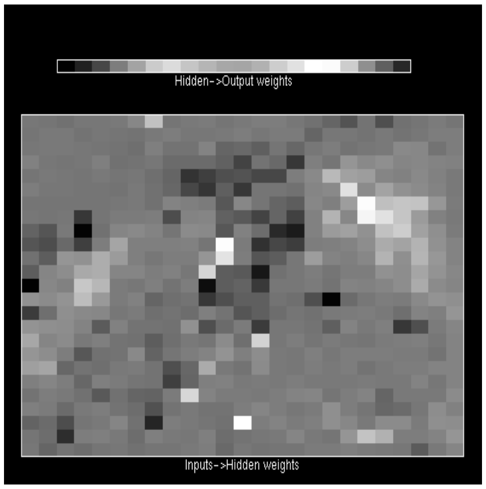

Figure 10: Hidden unit 2 weights

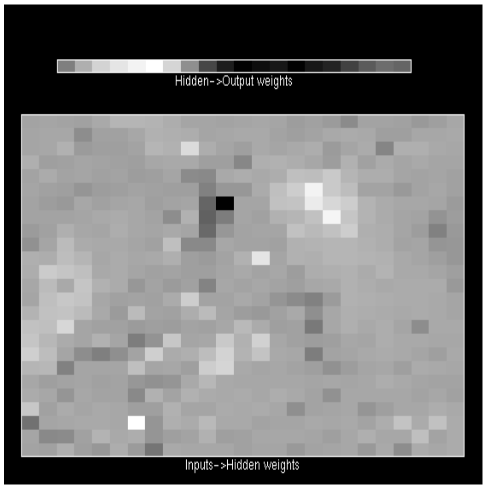

Figure 11: Hidden unit 3 weights

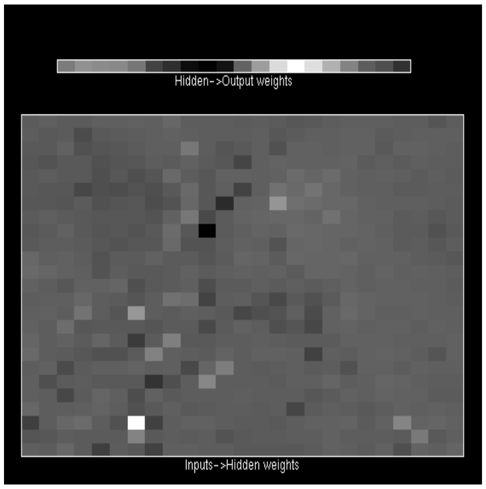

Figure 12: Hidden unit 4 weights

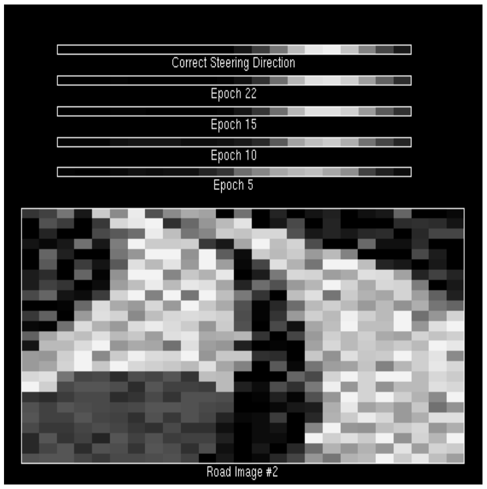

Figure 13: 2 lane road image

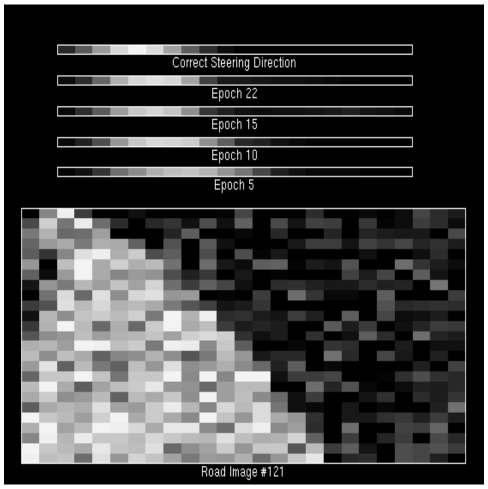

Figure 14: 1 lane road image

QuickProp weight diagrams

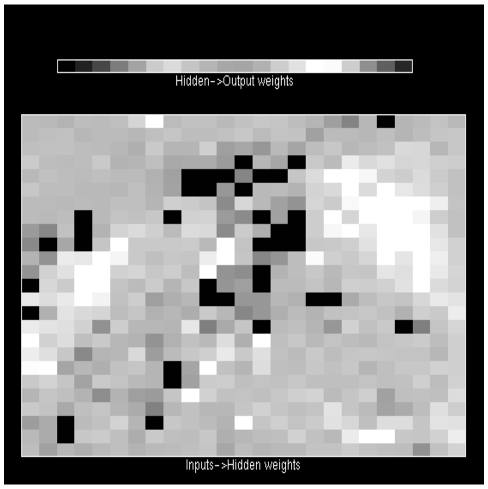

Figure 15: Hidden unit 1 weights

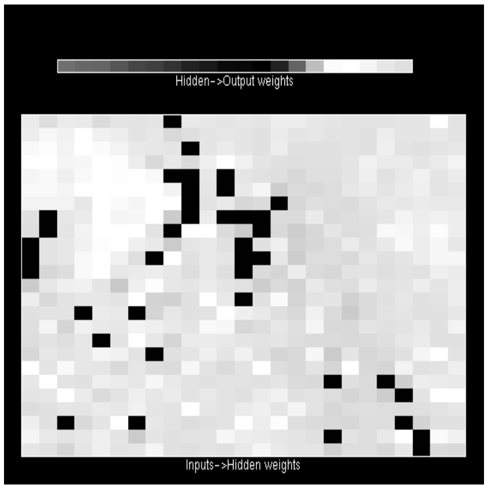

Figure 16: Hidden unit 2 weights

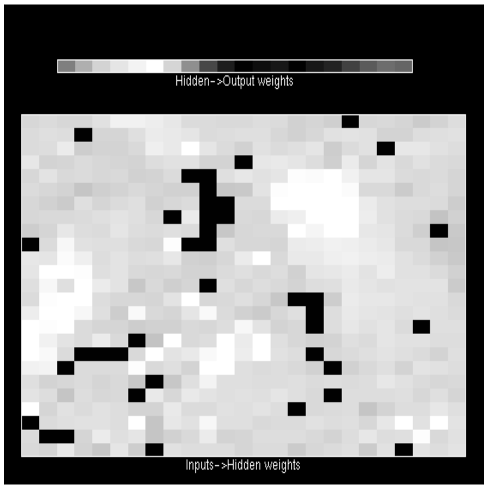

Figure 17: Hidden unit 3 weights

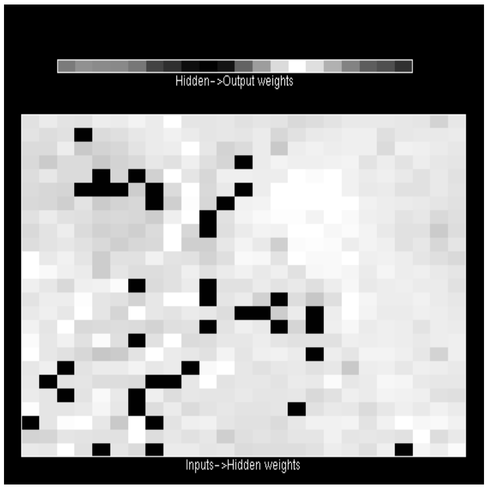

Figure 18: Hidden unit 4 weights

Enhanced QuickProp weight diagrams

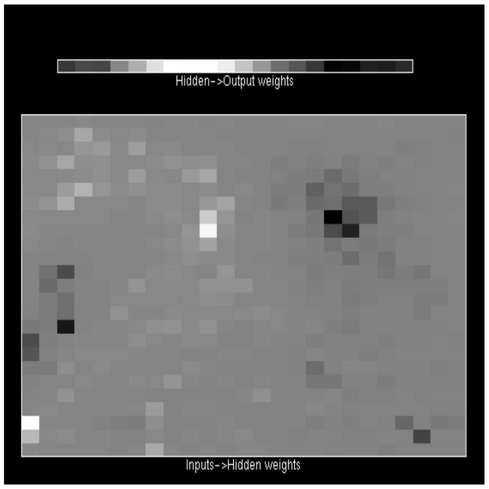

Figure 19: Hidden unit 1 weights

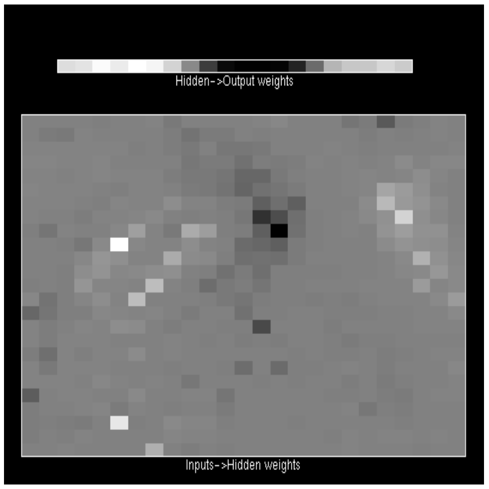

Figure 20: Hidden unit 2 weights

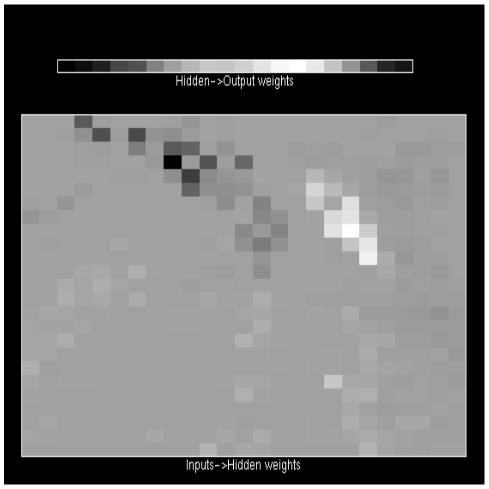

Figure 21: Hidden unit 3 weights

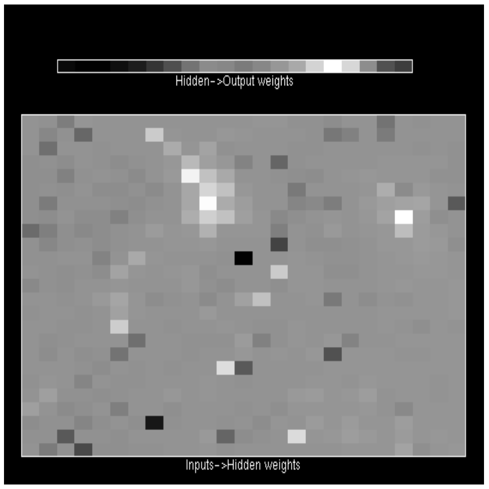

Figure 22: Hidden unit 4 weights

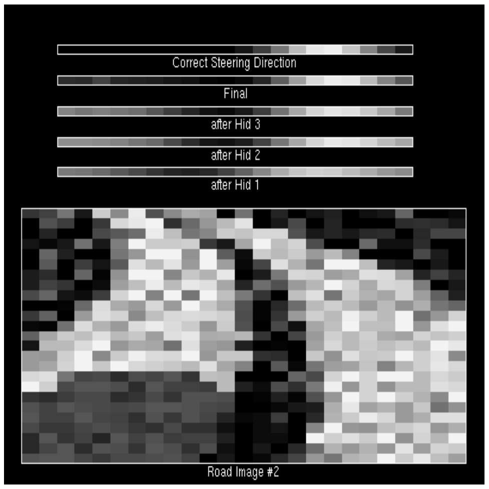

Figure 23: 2 lane road image

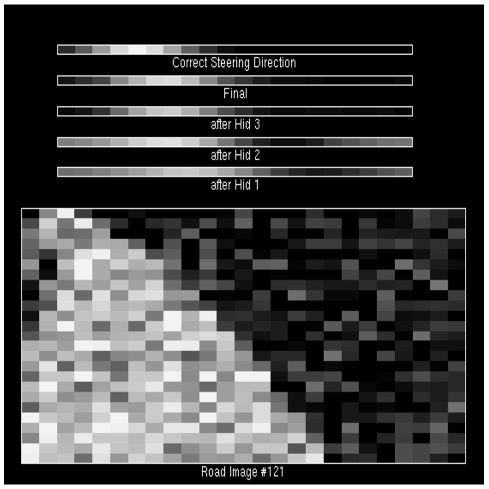

Figure 24: 1 lane road image

CasCor weight diagrams

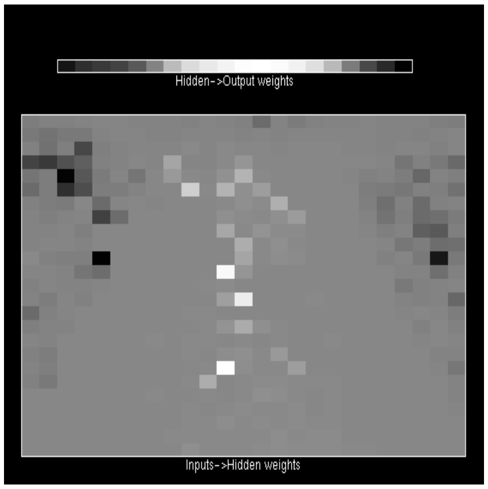

Figure 25: Hidden unit 1 weights

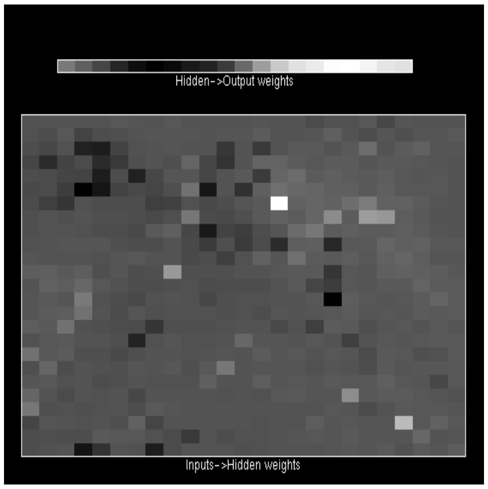

Figure 26: Hidden unit 2 weights

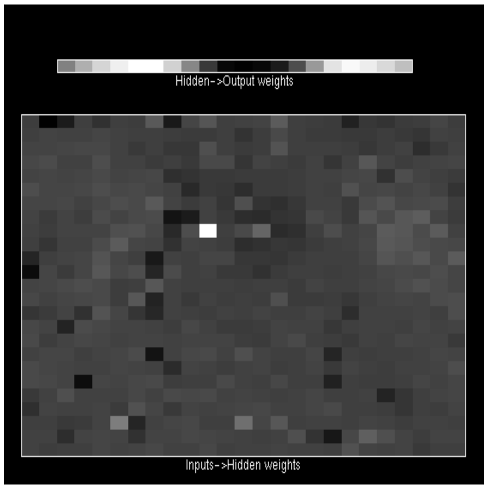

Figure 27: Hidden unit 3 weights

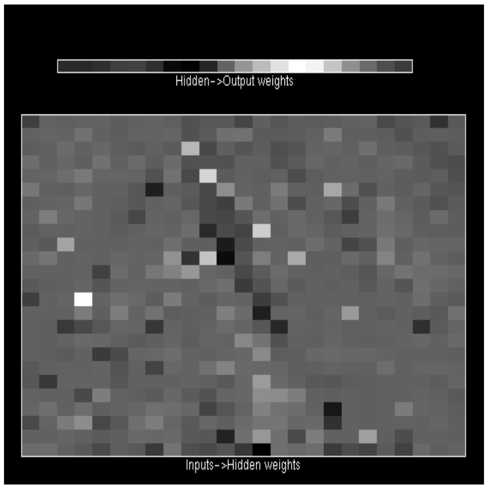

Figure 28: Hidden unit 4 weights

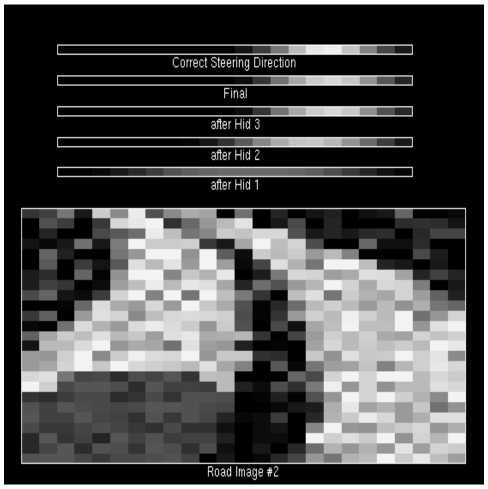

Figure 29: 2 lane road image

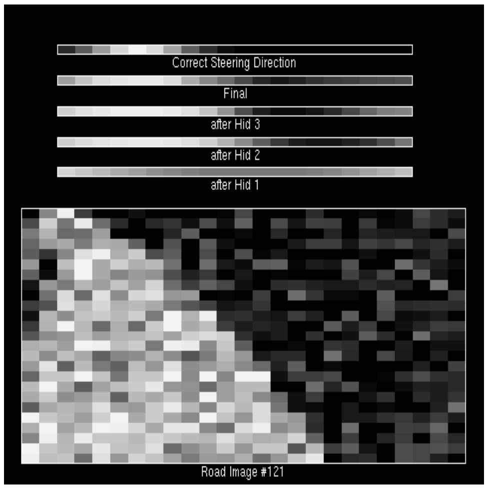

Figure 30: 1 lane road image

Cascade2 weight diagrams

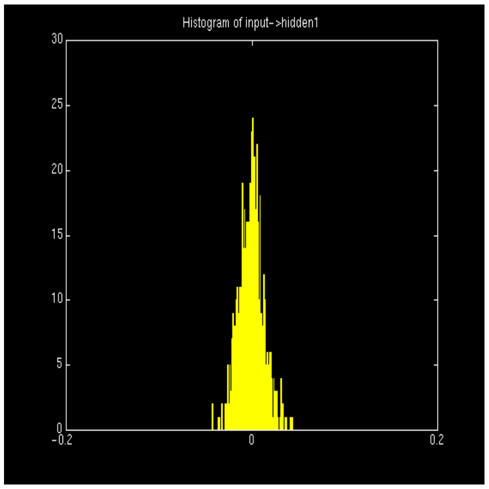

Figure 31: Hidden unit 1

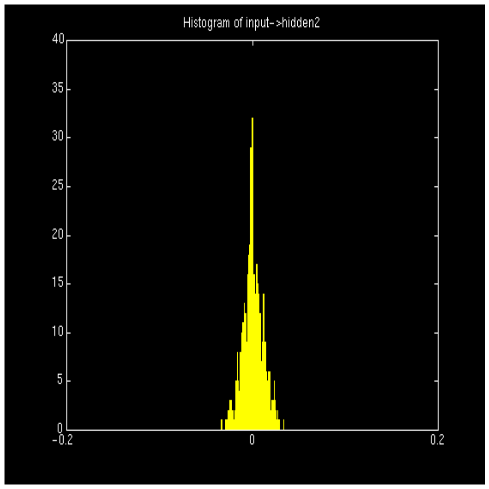

Figure 32: Hidden unit 2

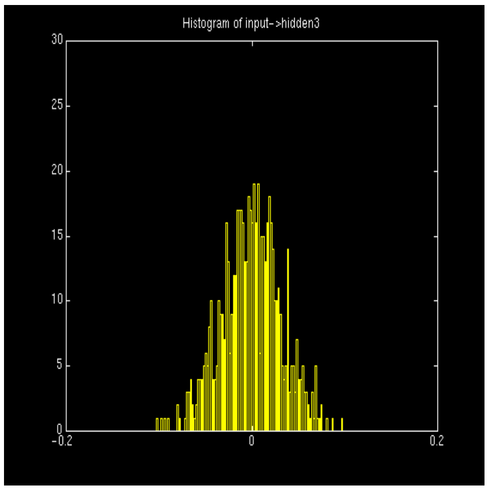

Figure 33: Hidden unit 3

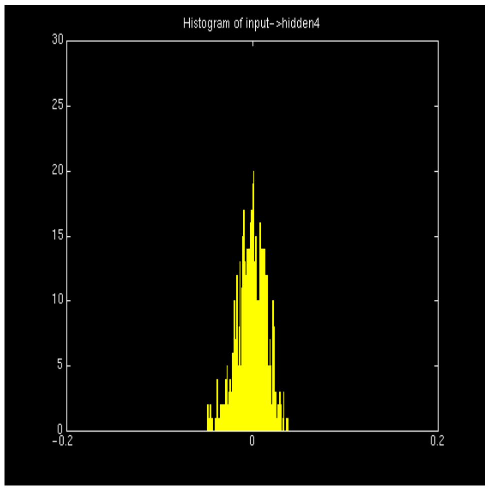

Figure 34: Hidden unit 4

BackProp histograms

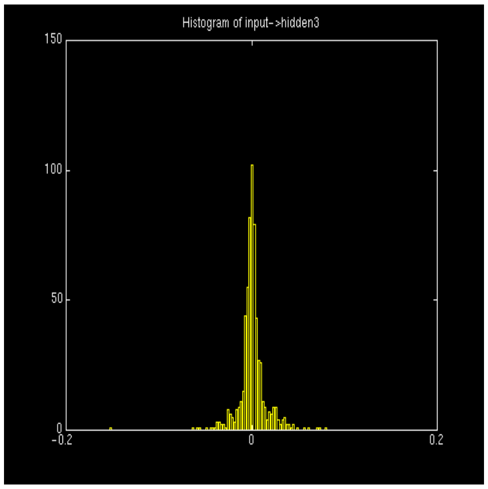

Figure 35: Hidden unit 1

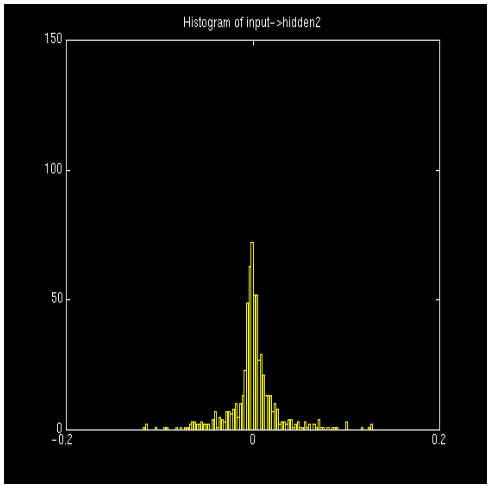

Figure 36: Hidden unit 2

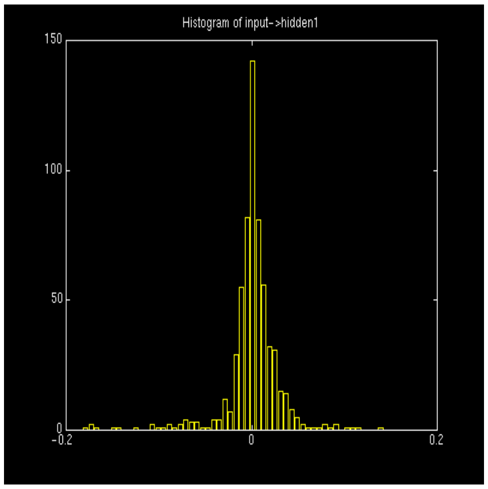

Figure 37: Hidden unit 3

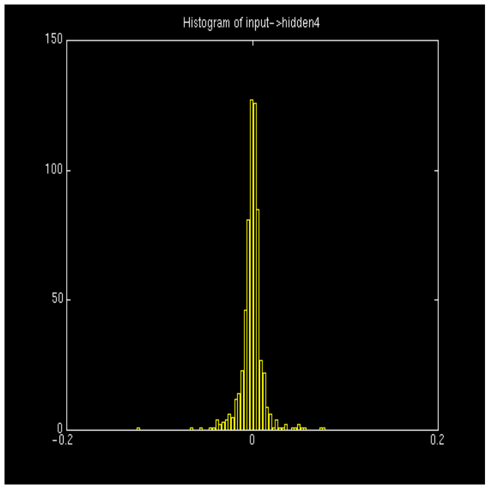

Figure 38: Hidden unit 4

QuickProp histograms

10. References

[1] Dean Pomerleau, "Neural Network Vision for Robot Driving", Robot Learning, Ed. J. Connell, S. Mahadevan, Kluwer Publishing, 1993.

[2] Dean Pomerleau, “RALPH: Rapidly Adapting Lateral Position Handler,” IEEE Symposium on Intelligent Vehicles, 1995.

[3] Mark Rosenblum and Larry Davis, “The Use of a Radial Basis Function Network for Visual Autonomous Road Following,” UMD Tech Report CAR-TR-666, 1993

[4] John Hancock and Charles Thorpe, “ELVIS: Eigenvectors for Land Vehicle Image System,” *CMU Tech Report CMU-RI-TR-94-43*, December 1994.

[5] Todd Jochem, Dean Pomerleau and Chuck Thorpe, “MANIAC: A Next Generation Neurally Based Autonomous Road Follower,” Proceedings of the International Conference on Intelligent Autonomous Systems: IAS-3," February 1993

[6] S. Fahlman and C. Lebiere, "The Cascade-Correlation Learning Architecture," CMU Tech Report CMU-CS-90-100, February 1990.

[7] S. Fahlman, "An Empirical Study of Learning Speed in Back-Propagation Networks," CMU Tech Report CMU-CS-88-162, June 1988.

[8] K. Knight, "A Gentle Introduction to Subsymbolic Computation: Connectionism for the A.I. Researcher," *CMU Tech Report CMU-CS-89-150*, May 1989.

[9] P. Bakker and J. Wiles, “An Animated Study of Learning Dynamics in the 14-2-14 Encoder,” U. of Queensland Tech Report 93-284, November 1993

[10] R. Wallace, A. Stentz, C. Thorpe, H. Moravec, W. Whittaker, T. Kanade, First Results in Robot Road Following, Proceedings of Int. Joint Conf. on Artificial Intelligence, 1985

[11] Maybeck, P.S., “The Kalman Filter: An Introduction to Concepts,” Stochastic Models, Estimation, and Control, Vol. 1, 3-16, 1979