MINNI: Micromouse Incorporating Neural Network Intelligence

Jondarr Gibb
Computing Department
Macquarie University
NSW 2109
jondarr@mpce.mq.edu.au

Len Hamey
Computing Department
Macquarie University
NSW 2109
len@mpce.mq.edu.au

Abstract

MINNI is a system whereby a back propagation neural network is used to control the steering of a micromouse (small robot) in following a straight path. The neural network must be minimised to run on the hardware/software platform available on the Macquarie Micromouse. The development of the network follows intensive trials of algorithm and architecture variations in MATLAB. *The implementation is programmed in* ADA.

Keywords Neural networks, robotics.

1 Introduction

The IEEE International Micromouse Competition runs events pitting robots from many universities at state, national, and international levels to promote the application of real time control in university computing and engineering departments. The *Rodentronics* team from Macquarie has known some success in the past.

The micromouse itself has undergone many changes over the years. Starting with the simple kit used as the basis of most competition entrants, controlled by C code, the group has designed and developed a more sophisticated model which now relies on an ADA run-time support system (Alsys 286 DOS Ada Vers 4.2) and an *Arcom Control Systems SCIM88* 80C188 CPU board. As far as is known, **MADAM** (the Macquarie ADA Mouse, figure 1) is the first of its kind, perhaps in the world.

This development has allowed the micromouse to be used in the teaching of an Advanced ADA course, wherein students program the robot to follow a simple path in the competition maze, as well as being used as the basis of some research projects. Normally, some form of feedback control technique is used to control the micromouse. Fuzzy logic has been applied to the steering control for cornering [11], but neural networks had not come into the picture.

Proceedings of the 20th Australasian Computer Science Conference, Sydney, Australia, February 5–7 1997.

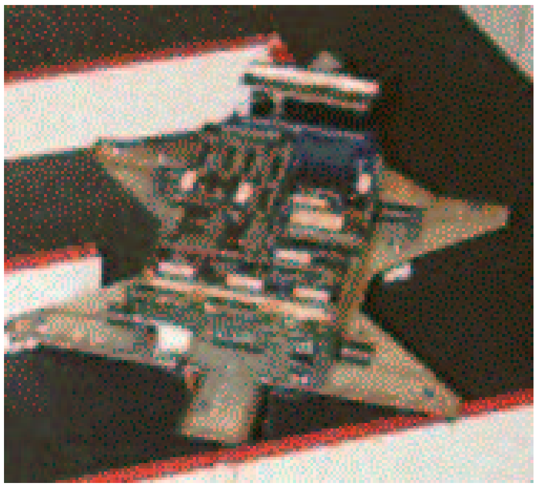

Figure 1: The Macquarie ADA Mouse.

Initially, the purpose of the research was to investigate the application of neural networks to control the micromouse when following a straight path. This is a non-trivial problem, due to factors such as variable slippage across the two driving wheels, differing motor wear (the wheels have independent control), initial orientation or position of the micromouse, and the varying sensitivity of sensors.

This project could be considered a scaled-down version of the ALVINN project [12, 8]. In that system, a large amount of computing power was used to steer a truck using video input. The training of that back propagation neural network required some filtering of the video signal, and manipulation of data patterns to extend the training set to gain generalisation (through modifying the input video data to indicate differently-shaped roads ahead). With sufficient computational power, the network was able to steer at quite high speeds and over great distances, reacting quickly to the environment. It holds the distance record for an autonomous vehicle on a public road.

This research will be using simple sensors instead of video input, and relying only on the immediate position of the robot relative to the path, rather than the shape of the road ahead. A back propagation neural network is used for simplicity of implementation, requiring very little

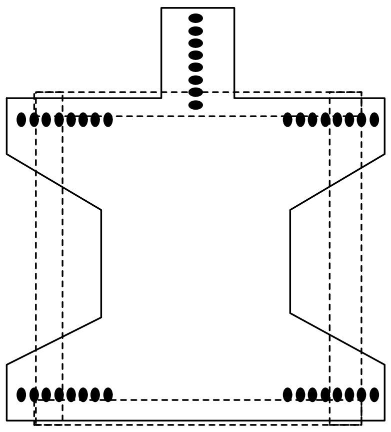

Figure 2: The layout of the sensors on the mouse.

memory or computational power from the limited resources available on-board the robot. This seems to be more in the spirit of the amateur nature of the micromouse competition as well as being 'sufficient' to solve the problem. A more complete discussion of the research project can be seen in Gibb [4].

2 The Micromouse

In the simplest case, the micromouse will travel between two walls, and the network is expected to steer the micromouse based on the steering performed by standard programming techniques, trying to keep it on a straight path between the walls.

The mouse steers by sensing the red-coloured tops of white walls mounted on a black base board, using red-detect sensors in banks of eight in a configuration shown in figure 2. The dotted lines show the approximate location of walls that might be met including a wall behind the micromouse – the starting location, and one in front – the stopping condition. In a competition maze, gaps in the walls through which the mouse might turn. Steering is achieved by varying the speeds of the two motors powering a wheel each, centrally located on the micromouse.

Although the initial research used all four sensor banks, the final version was trained on only the left-hand banks. This served two purposes: it made the resultant program smaller, as it had fewer weights in the network; and it also made the small data sets more relevant for training in cross-validation. No significant difference between the two- and four-sensor networks was seen during training in terms of the shape of the learning curve, and therefore training time.

3 The Network

The output of the net is a steering direction value between -7 (hard left) and +7 (hard right), which (non-linearly) controls the relative speeds of the

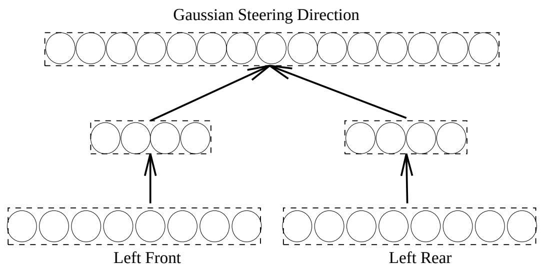

Figure 3: The neural network incorporating left side only.

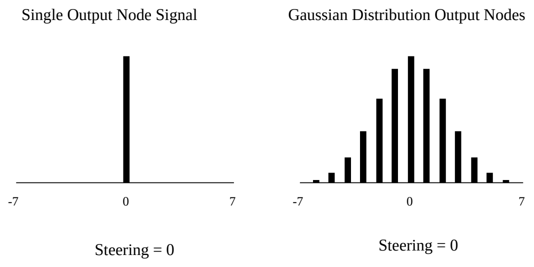

Figure 4: Example steering output of the network.

motors on the micromouse, thus forcing it to veer off in a new direction. This is a low-level software control criterion.

In theory, the left hand sensors alone should be sufficient to steer the micromouse when it is guaranteed that there are no gaps in the wall being followed, and so only the left front and rear sensor banks are needed as input to the neural network. This gives us a simple network as shown in figure 3, where the heavy-arrowed inter-layer connections represent full connectedness.

The hidden layer configuration of four hidden nodes for each sensor bank is used to determine features of that sensor bank relevant to overall steering. This might be considered a local decision in feature detection.

There is a natural tendency to align left-oriented sensor bank inputs with a leftward steering direction (that is, the mouse is drifting away from the left wall), and similarly the converse. It was seen that something akin to this is done by the network. This suggests that the hidden layers 'summarise' the input tendency. The weighted connections between each of the two hidden node groups and the output layer should therefore simulate a decision on how much each of the two local decisions will contribute to the resultant steering direction.

Variations on this architecture were investigated, changing the number of hidden nodes, and adding (or removing) connections between the sensor banks and the hidden layer.

3.1 Output Specification

Although the selection of one output over all others is a requirement of the system (as designed), so as to give a discrete steering direction, it is often difficult to train a network to distinguish strongly between subtly different nearby categories. This can be seen in figure 4, where a single output node on the left side becomes a sequence of graded outputs on the right. Pomerleau [8] shows how the difference between the binary output patterns representing a direction of 0 (straight ahead) and -1 (just to the left), is quite dramatic, and therefore might be difficult to train the network to handle.

On the other hand, using a gaussian response in the output nodes to approximate the steering direction gives a minimal change between neighbouring output patterns.

3.2 Data Requirements

For good generalisation, the data must sufficiently cover the range of possible input-output pairs that the micromouse will see, including extreme conditions of a sharp left- or right-hand turn. The data must also be sufficient in number so that the subset presented to the network does not cause over-fitting of the solution due to the number of free parameters (weights) in the network.

It is a generally-held principle that the number of patterns in batch-learning for back propagation should exceed ten times the total number of interconnecting weights and biases (free parameters) in the network. There are $2 \times 8 \times 4 + 2 \times 4 \times 15$ connecting weights, plus $4 + 4 + 15$ bias weights (total of 207 connections). However, as there is effectively one analogue output node, there are, in effect, a lot fewer free parameters in question. This would imply that 1000 patterns is more than enough to train this network without worrying about over-fitting.

Many investigators, such as Jackel [6] and Amari [1] show details of evaluating the size of a data set needed for various problems.

3.3 Data Collection and Creation

The initial approach to training was to use the output from a collection of programs written in Ada for the micromouse by various postgraduate students. These programs were developed independently (from a common base of lower level control) and at various levels of complexity. The authors added their own to this collection of programs.

Each of these programs was run on the micromouse, which was put at various start locations relative to the two fixed straight lines of red-topped walls. Various velocities were used to give a greater variety of required steering to successfully follow the walls.

However, this collection is not sufficient. No single run of the micromouse generates more than 100 patterns, and it is not necessarily reasonable to string together different programs' outputs to form one training set, as this might introduce conflicting input-output pairs. Variations within one program's successive runs may not encompass enough pattern possibilities.

One way of getting a sufficient data set is to create one artificially, using an algorithm which determines a steering direction for each member of a collection of all possible sensor bank inputs.

Many data sets were created in this manner. One set used all four sensor banks to determine the steering direction, after which the right-hand banks were removed, along with input-output pairs which could not be learned (due to the fact that they had no input data). Another set used only the two left-hand sensor banks to make the decision. Extracts of up to 1000 patterns from these *supersets* were obtained by statistically representing the expected inputs with a gaussian distribution centred on the straight-ahead steering direction.

Only small variations in training were seen across these data sets, including a randomised order set, as the patterns were generated in order from a left-hand steering output to a right-hand one. Added to these artificially-produced data sets were several small runs collected directly from the mouse, in particular a *good* run which allowed the mouse to slowly drift from right to left, and a particularly *bad* run which displayed oscillatory steering behaviour.

4 Learning Platform

Regardless of the network being implemented (architecture, algorithm, etc), a reliable platform must be found to both train the network and simulate the running of the application (the micromouse) before the real-world situation is met.

The data collection was achieved with Ada programs running on the micromouse. The final outcome of the project was the production of an Ada implementation of a back propagation simulator based loosely on the C code associated with Fahlman's *quickprop* demonstration software [2].

It is not expected that the micromouse would need to be retrained at any stage, as the situation is such that the operation of following a wall will not change considerably over time, and therefore will not need continual adjustment. The variation in motor wear and wheel slippage will be ignored, as the network should be able to handle this situation (within limits). A set of weights will be found to suit the problem, and that will be hard-coded into the feed-forward neural network Ada code.

Choice of software for developing a neural network solution for any problem is an important issue to consider, and should not be based more on personal preference than computational reliability. Following the work of Gibb and Hamey [5], it was decided that the reliable platform on which to develop the network solution would be MATLAB. This allows control of the network's learning in a user-friendly environment from which a sequence of weight initialisations can be extracted easily to insert into the awaiting ADA program.

5 Learning Results

Learning a solution to the problem through training the network is but a small part of the implementation. Through trialling algorithms, we hope to learn something about the problem itself which may be useful in finding a way to a solution, even if it does not yield a good one. Many variations on the standard back propagation algorithm were used in an attempt to fully investigate the data set and find the best solution for the problem – to produce a good set of weights which could then be hard-coded into the ADA program.

Varying the fixed value of both learning rate and momentum showed effects on the stability and speed of learning. A value for momentum of 0.5 was found to be useful in most instances, but this was varied to investigate the effects thereof. Smaller momentum, near 0, caused instabilities in the learning, whereby the network was caught in small local minima easily, or sometimes increased the error of the network. However, larger rates (as high as 0.9) restricted the learning, often missing the useful ravines which would lead to better minima. A small value for learning rate, in the range [0.01..0.2], seemed appropriate to avoid large oscillations in the learning. Values for learning rate over 0.4 stopped the network from converging to a satisfactory level, never letting it settle down.

5.1 Cross Validation

Cross validation applied to neural networks, as described by Sarle [9], and Sjöberg and Ljung [10], is one way of seeking generalisation and avoiding overfitting of the data, which may occur when the number of free parameters is large. Due to the nature of the problem, the training set used was artificially created, while the cross validation (CV) set was a collected one. This allowed the generation of as much data as was needed, along with the ability to relate the solution back to the real-world problem through careful monitoring of generalisation.

This technique was employed when training the network, producing two significant scenarios: one in which the error on the CV set was minimised

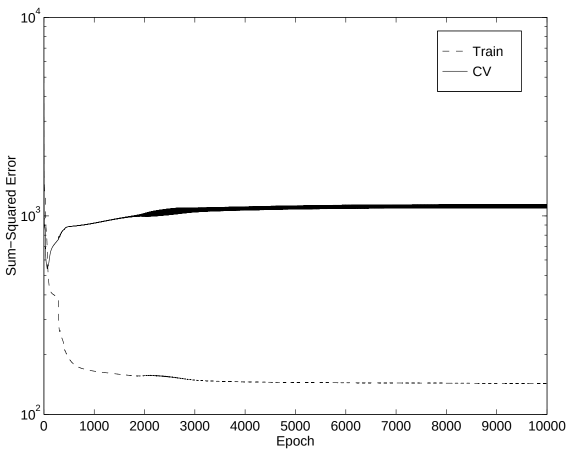

Figure 5: Training and Cross Validation learning, early minimum.

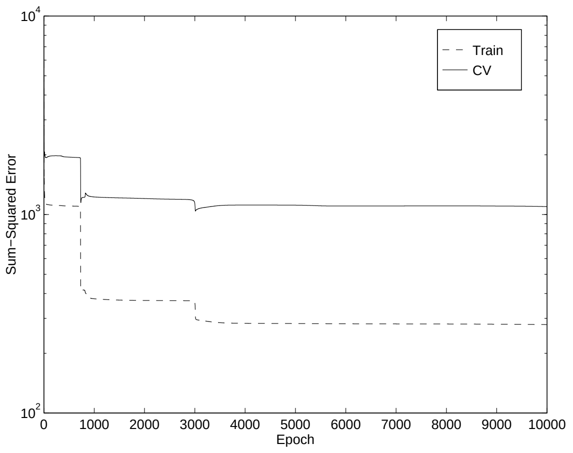

Figure 6: Training and Cross Validation learning, following.

early in training (after very little learning had occurred), as shown in figure 5, and the other where the CV error had no discernible minimum before training was halted by a hard limit, such as in figure 6.

Oscillatory behaviour (figure 5) was common, and this indicates the inherent differences between the two data sets used. After only a short period of learning, further minimisation of the training set error is detrimental to the CV set error. The training set error-weight space could be described as mostly orthogonal to the CV error-weight space, as further small variations in the training set error cause such high-frequency oscillations, implying entrapment of the network in a multi-dimensional valley. For the training set, this valley leads to the error minimum, and travelling along its floor is beneficial. However, the small oscillations in the weights are multiplied in the CV error-weight space, as the network is not travelling along the floor of the valley, but on a slope.

On the other hand, figure 6 gives a situation where it is not obvious where the CV error mini-

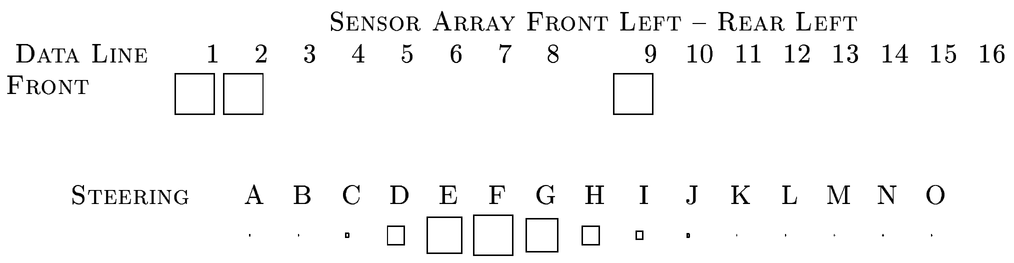
SENSOR ARRAY FRONT LEFT – REAR LEFT
DATA LINE 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
FRONT
STEERING A B C D E F G H I J K L M N O

Figure 7: Sensor inputs to Steering outputs (3 hidden nodes).

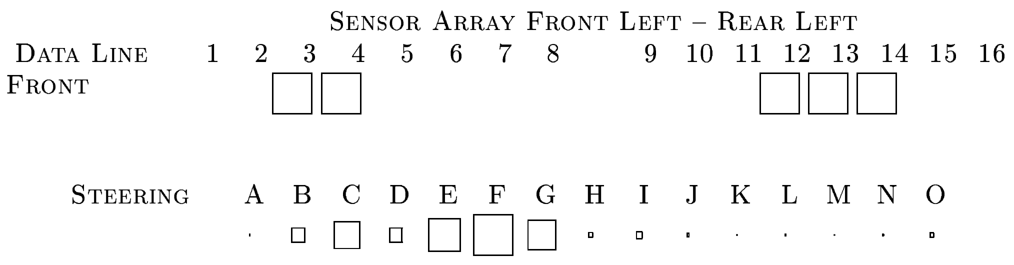
SENSOR ARRAY FRONT LEFT – REAR LEFT
DATA LINE 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
FRONT
STEERING A B C D E F G H I J K L M N O

Figure 8: Sensor inputs to Steering outputs (5 hidden nodes).

mum should be. Further training may illicit further drops in the CV error (this is true, but they come less frequently). It is important to note that the current minimum in the figure occurs at the point of the last drop. This is a significant point – each drop in training set error has caused a corresponding drop in CV error (they are highly correlated) until over-fitting occurs, from whence the training set error is still dropping (at a different rate), and the CV error begins to rise for some time before slowly falling again. It is usually the case that the CV error falls below its previous minimum before the next significant drop in training set and CV set error occurs.

Further discussion on the use of cross validation and artificial data in problems such as this can be seen in Gibb [3].

5.2 Architecture Variations

The architecture chosen was $2 \times (8 \rightarrow 4) \rightarrow 15$. Variations on this decision showed that the initial choice was a reasonable one. Different numbers of hidden nodes were trialled (0, 2–6), and both fully- and partially-connected networks were trained. The overall result was as expected, that is, the smallest networks (0 or 2 hidden nodes) did not learn sufficiently well, the network with 3 hidden nodes did not generalise, and networks with 5 or 6 hidden nodes over-fitted the data.

A variation on Hinton diagrams is used to show the input/output signature of the network, where the input sensor banks show white squares to indicate that a sensor is on (detecting the wall beneath it), and the output steering direction should be described by a gaussian curve of node strengths across the output nodes.

The result of the 3 hidden node network can be seen in figure 7, where the inputs would indicate a rather sharp left-hand turn, while the outputs of the network show only a slight deviation. It is important to note that the shape of the gaussian is near perfect. On the other hand, when there are 5 hidden units, as in figure 8, and only the front left sensors indicate that a turn should be made, there is some indecisiveness in the steering direction chosen, causing a badly-shaped gaussian output.

A good example of the way in which the network weights are used to create the output gaussian direction is given in the Hinton diagram of figure 9. This shows positive weights as white, negative as black, of an area representing the relative size of the weight. As can be seen, sequences of output nodes are attached to one hidden node to form small ramps and humps, usually in positive and negative symmetric pairs. These approximate the derivative of the gaussian curve, combinations of which form the gaussian output steering direction dependent on the strength of each of the hidden nodes' outputs.

6 Network Testing

Testing was performed by simulating the network under Unix using a series of simple text-file reading stubs to emulate the sensor inputs. This method indicated that the best set of network weights was obtained from those training runs where the cross validation error for a non-related data set did not

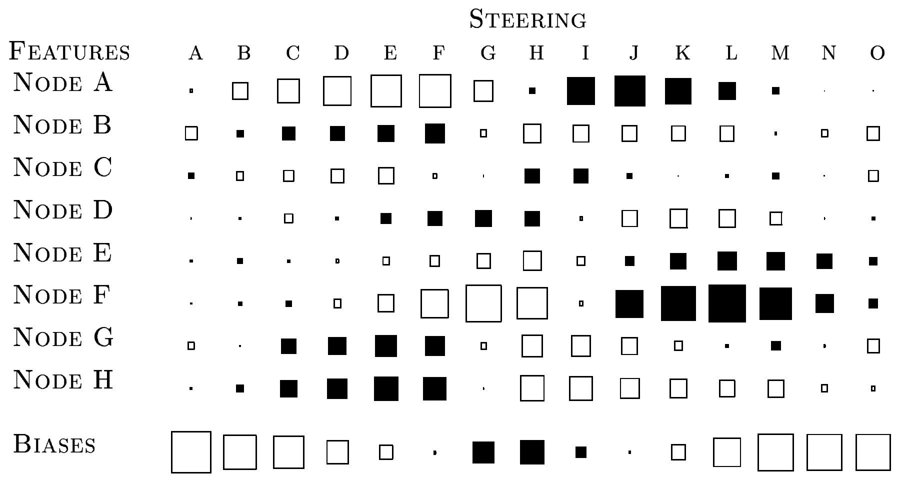

Figure 9: Fully-connected network output layer weights.

reach its minimum too early in the learning process. When an artificial data set was being trained on, then one of the collected micromouse runs was used as the CV set.

Not only did this produce a network with a high correlation to the running of the micromouse by the other (successful) traditional programs, but that network performed better than with any other set of weights obtained.

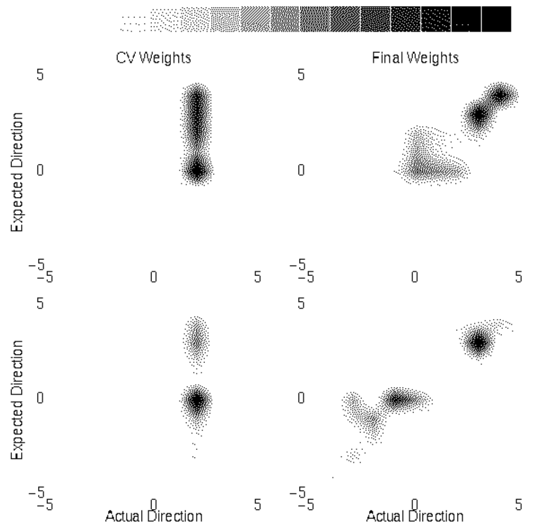

Figure 10: Correlation Comparison for Cross Validation Training

Looking at figure 10, the correlation be seen. This diagram represents the amount by which the network's output correlates with that of the program which produced the CV data set. Darker regions show a higher correlation. What is expected is that the dark regions lie mostly along the diagonal – that is, where the *actual direction* chosen by the network aligns with the *expected direction* given by the program. There are two distinct trials here (upper and lower half), and two sets of weights (left- and right-hand side). The first set of weights represents a situation where the cross validation error reached a minimum very early in the learning process, and so the left-hand network's weights were frozen at that point; whereas the right-hand side network's weights are those obtained when training was not ceased before some pre-specified epoch maximum (10000 epochs).

As can be seen, the learning achieved by the left-hand side network is not sufficient to run the micromouse. The direction chosen is invariably around *2* for all situations. On the other hand, the results of the right-hand side network are far more encouraging. In this case, a wide diagonal band can be seen to represent areas in which the network and the program correlate.

It should be noted that these weights are obtained where the final CV error is higher than its maximum (such as in figure 5). In training situations such as that seen in figure 6, where the maximal epoch is also the point of lowest CV error, an even higher correlation was found, with less variation from the main diagonal.

A discussion of the use of CV sets and artificial data, and measuring the performance of multi-output networks is given in Gibb [3].

6.1 Micromouse Performance

Unfortunately, the difference between a simulation of the micromouse in a comfortable Unix environment on a desk and a chunk of electronics propelled along a red-lined black channel by two independently motorised wheels and a chassis of rattlingly-overloaded aluminium is not nominal. Having seen in theory how close the network's output is to any other program used to run the micromouse, it was necessary to run the ADA program *in situ* to see how well it negotiated a real path.

Speed became an issue, as there is a limit to how slow the mouse can be run effectively before it stops under its own weight. For instance, the code which fits the best gaussian curve to the output of the network, to obtain a steering direction, is quite compute intensive, as it finds the least squares difference between proposed curves. At first, this method was discarded for a simple algorithm which found the first peak and pretended that that was the gaussian centre. This was never the case, and simple experiments showed that the software performed poorly by comparison to the original in spurious cases, thus lowering the overall performance.

When compared to the original, simpler program written in ADA, there was always going to be a speed difference. It was found that the decision-making process of a neural network implementation slowed the micromouse to such an extent that it could safely travel at only 75% of the speed it could before. This went up by almost 5% when the gaussian curve fitting was removed. This comparison is for success in most cases, that is, as the speed increased, it was less likely to deviate from the straight line, but when it did, the results were somewhat catastrophic.

This slowing down of the process goes against the grain of neural network implementations in general, whereby the categorisation of inputs is often more efficiently done by the neural network than 'manual' manipulation of the input. In this case, however, the original control program performed only a simplistic, and yet mostly effective calculation to give its steering direction.

Repeated testing for reliability showed that the micromouse had started listing severely, and that it travelled in an arc to the left when intended to go straight ahead (without the use of any control software). This caused problems in the testing, in that it became a likely situation that the micromouse would slowly drift to the left-hand wall, under directions from the neural network to steer a small degree to the right with no result. This often left the micromouse so close to the wall to be touching it at both the front and rear, and thus unable to turn away from it. This situation occurred approximately every 18-19 squares of traversal (with great inconsistency due to initial orientation and positioning of the micromouse). The maximal length of the path used was 11 squares, and the maximal competition straight path is 16 squares.

7 Conclusions

This project shows that ADA, as a language, is not so unwieldy or inappropriate either in neural network or small real-time control applications. It also shows that computing power and a lot of data are not necessary to control a simple robot.

The analysis of the problem of controlling a micromouse's steering through the use of a back propagation neural network implemented in ADA has overcome many difficulties along the way. Complexities have arisen in the production of the resultant program and in the way in which the search for a solution to the general problem proceeded. Although some of these difficulties associated with the data set could possibly have been alleviated by the use of filters applied to the input or output training patterns, the purpose of the research has always been to implement a simple solution to the problem to be run in real time on the available hardware.

The use of cross validation as a means of selecting a particular training instance, and resultant network, over others, in general terms of the shape of the learning curve, has been investigated. Where artificial data is involved in the process of training, then it is imperative that the network's learning be in some way related back to the original application through an injection of real-world data checks and balances. Cases where training is suspended due to the network being caught in local error minima do not sufficiently solve this problem. Even when local minima for the training data do not affect the training, a solution may not be found when the error surface does not relate to the real-world data patterns for which the network is proposed.

However, cross validation cannot be applied in an *ad hoc* manner, as situations where the error on the cross validation set oscillates wildly or bears no correlation to the training set error during training give almost useless resultant weights. The way in which the network trains is important to the selection of the network used in an application. In theory, the choice of a particular algorithm is usually done on the basis of expected generalisation, or some such measure, by averaging many network training runs. It should always be remembered that the final application will have only one set of weights in the neural network, and this must be obtained from one (successful) run.

It is important that this singular choice is based on more than theoretical superiority of a training method when initial conditions and training data may bias the result so noticeably (as discussed by Kolen and Pollack [7]).

Following in the footsteps of Pomerleau [8], this research has come in sight of that goal of neural network control of an *autono-mouse* vehicle. With less computing power available (an 80C188 versus a Warp engine), less foresight (current orientation of the micromouse versus visual analysis of the road ahead), and less filtering (no image processing), the micromouse has been taught not only to follow a straight line (as envisaged), but practically any gently curved path bordered by the appropriate red-topped white walls of a maze structure.

The bare minimum has been employed at each stage of the research. As far as network architecture is concerned, only two of the four sensor banks have been used for the neural network, with an optimal/minimal hidden layer. The training data has numbered only what is considered a necessary requirement for training a neural network of that size, and collected data patterns have been used for verification. The algorithm used was standard back propagation (although many variations were investigated), but the choice of resultant weights has been carefully implemented. The hardware and software used for implementation are rudimentary, incorporating no great computing power or programming efficiencies, except through minimising the network implementation (to remove training, for instance).

References

[1] S. Amari, N. Murata, K.-R. Müller, M. Finke and H. Yang. Asymptotic staistical theory of overtraining and cross-validation. Technical Report METR 95-06, Dept Mathematical Engineering and Information, University of Tokyo, Hongo 7-3-1, Bunkyo-ku, Tokyo 113, Japan, August 1995.

[2] Scott E. Fahlman. An empirical study of learning speed in back-propagation networks. Technical Report CMU-CS-88-162, Carnegie Mellon University, Pittsburgh, PA 15213, September 1988.

[3] Jondarr Gibb. Cross validation and performance measures in training on artificial data. (to appear), 1996.

[4] Jondarr Gibb. Generalisation and performance of a back propagation network applied to a small robot. Master's thesis, Dept Computing, Macquarie University, 1996. (submitted).

[5] Jondarr Gibb and Leonard Hamey. A comparison of back propagation implementations. Technical Report C/TR95-06, Computing Dept, Macquarie University, 1995.

[6] Larry Jackel, et al. Practical applications of capacity control to neural net learning. Invited Presentation, ACNN95, February 1995.

[7] John F. Kolen and Jordan B. Pollack. Backpropagation is sensitive to initial conditions. Technical Report 90-JK-BPSIC, Computer and Information Systems Department, Ohio State University, Columbus, Ohio 43210, 1990.

[8] Dean A Pomerleau. ALVINN: An autonomous land vehicle in a neural network. PhD thesis CMU-CS-89-107, Carnegie Mellon University, Pittsburg, PA, 1989. Book format only version now available.

[9] Warren S. Sarle. Tweakless training: Neural network training benchmarks using sas software. draft, SAS Institute Inc, SAS Campus Dr, Cary, NC 27513, January 1994.

[10] Jonas Sjöberg and Lennart Ljung. Overtraining, regularization, and searching for minimum in neural networks. Technical Report LiTH-ISY-I-1297, Department of Electrical Engineering, Linköping University, Linköping, Sweden, May 1992.

[11] Teck Wah. Robot control with ada and fuzzy logic. Honours thesis, MPCE, Macquarie University, 1993.

[12] Richard Wallace, Anthony Stentz, Charles Thorpe, Hans Moravec, William Whittaker and Takeo Kanade. First results in robot road-following. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Volume 1, pages 1089–1095, 1985.