To appear in the May 1995 issue of "Simulation" - journal of the Society for Computer Simulation.

Neural Network Control of a Free-Flying Space Robot

Edward Wilson* Stephen M. Rock†
Stanford University
Aerospace Robotics Laboratory
Stanford, California 94305
ed,rock@sun-valley.stanford.edu

January 25, 1994

Abstract

Two recent developments in neural network control are presented. First, a "Fully-Connected Architecture" (FCA) is developed for use with backpropagation (BP). This FCA has functionality beyond that of a layered network, and these capabilities are shown to be particularly beneficial for control tasks. A complexity control method is applied successfully to manage the extra connections provided, and prevent over-fitting.

Second, a technique that extends BP learning to discrete-valued functions is presented. This algorithm is applicable whenever a gradient-based optimization is used for systems with discrete-valued functions. The modification to BP is very small, simply requiring replacement of the discrete-valued functions with continuous approximations and injection of noise on the forward sweep.

The viability of both of these neural network developments is demonstrated by applying them to a thruster mapping problem characteristic of space robots. Real-world applicability is shown via an experimental demonstration on a 2-D laboratory model of a free-flying space robot.

*Ph.D. Candidate, Department of Mechanical Engineering. Research partially supported by NASA and AFOSR.

†Associate Professor, Department of Aeronautics and Astronautics.

1 Introduction

Due to their capacity for adaptation, learning, and very high computational throughput, neural networks are promising for control applications. In some cases their learning abilities and inherent non-linear nature allow them to solve control problems and provide performance unmatched by conventional methods. In other cases their distributed nature and resulting computational power allow them to implement known solutions more quickly and robustly than conventional serial processors.

Neural networks derive their advantage in solving very complex problems from the emergent properties that come with the massive interconnection of simple processing units. With good training techniques, the networks are capable of implementing very complex behaviors. For example, neural networks may be used to implement arbitrary mappings of inputs to outputs, such as from sensor signals to actuator commands in a control problem. Further, since the mapping can be taught indirectly, neural networks are especially attractive for poorly-understood systems - they can generalize from training inputs and then respond in untaught situations.

Due to the distributed nature of the processing, networks are often robust to internal component failures; the remaining processors can adapt to account for the failure. Similarly, the network can be made to adapt to changes in the environment, plant, performance criteria, etc.

These features of neural networks make them particularly attractive for control applications. There are numerous examples in the literature that demonstrate the potential of neural networks in this area. See for example [1].

There are two issues which often arise in a real-world control application that have not been effectively addressed in the NN literature, however.

- *A priori* knowledge is often available in the form

of a preliminary control design (e.g., provided by "conventional" control design techniques). It should be possible to use this a priori knowledge to improve the learning performance of the network.

- Many control applications involve the use of discrete-valued devices. For example, thrusters often operate "on-off" rather than continuously. This presents a problem for backpropagation learning since these functions are not continuously differentiable.

The goal of the work reported here was to develop extensions to neural network theory that would address each of these issues. In particular,

- A "Fully-Connected Architecture" (FCA) was developed to allow direct input of an *a priori* linear solution.
- A general method of Backpropagation Training with "Noisy Sigmoids" was developed to allow learning with discrete-valued functions, such as the on-off thrusters.

In order to demonstrate the application and utility of these extensions, an experimental demonstration was performed. The task was to develop a neural network controller that would control the (x, y, θ) position and velocity of a free-floating robot using on-off thrusters. The apparatus is shown in Figure 1.

In Section 2, this experimental equipment (i.e., the robot) is described in more detail, and the particular thruster mapping problem addressed is presented.

In Section 3, the "Fully-Connected Architecture" is presented. It is used with backpropagation (BP), and is shown to have greater functionality than a standard layered network. Particular benefits of the FCA, some of which are especially useful for control problems, are outlined.

In Section 4, a method is presented that allows BP learning with systems containing discrete-valued (and therefore not continuously differentiable) functions (such as the on-off thrusters). This method requires only simple modifications to standard BP, and extends to multiple layers of hard-limiting neurons or the FCA with no need for modification.

In Section 5, the results of applying the FCA and the modified learning to the experimental apparatus are presented. The results verify the viability of the techniques.

2 Robot Control Application

The example control task addressed in this research is the control of position and attitude of a free-flying space robot using on-off thrusters. Control using on-off thrusters is an important problem for real spacecraft, and the non-linear and adaptive capabilities of neural networks make them attractive for these problems. Additionally, a NN-based method is likely to scale well to higher dimensional thruster controllers, as well as providing a structure conducive to reconfigurable control.

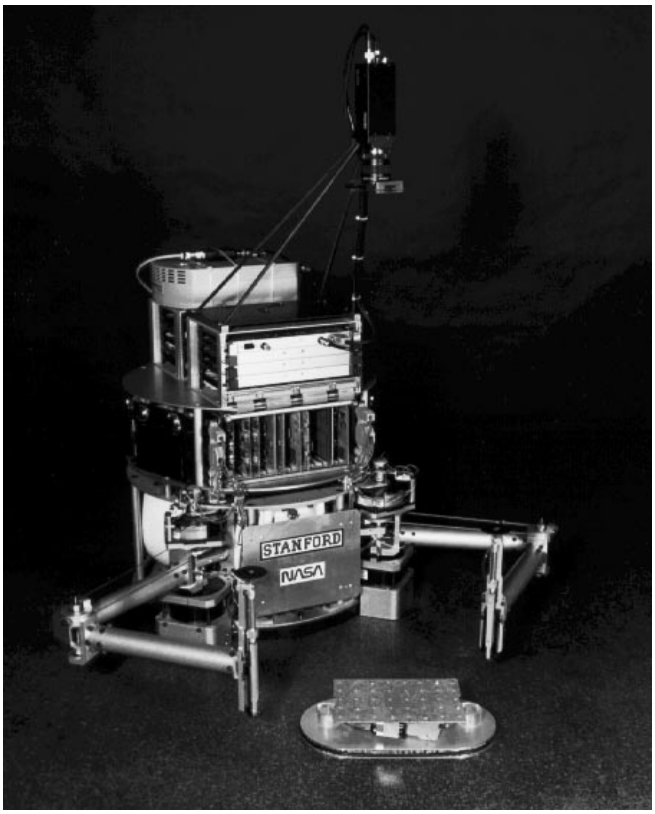

Figure 1: **Stanford Free-Flying Space Robot**

The experimental equipment used here to represent this problem consists of a mobile robot that operates in a horizontal plane, using an air-cushion suspension to simulate the drag-free and zero-g characteristics of space. This robot, shown in Figure 1, is a fully self-contained planar laboratory-prototype of an autonomous free-flying space robot complete with on-board gas, thrusters, electrical power, multi-processor computer system, camera, wireless Ethernet data/communications link, and two cooperating manipulators. It exhibits nearly frictionless motion as it floats above a granite surface plate on a 50 micron thick cushion of air [2].

2.1 Thruster Mapping Problem

The three degrees of freedom (x, y, θ) of the base are controlled using eight thrusters positioned around its

perimeter, as shown in Figure 2. The on-off nature of the thrusters substantially complicates the control design, due to their discontinuous nature and the fact that each thruster simultaneously produces both a net force and torque.

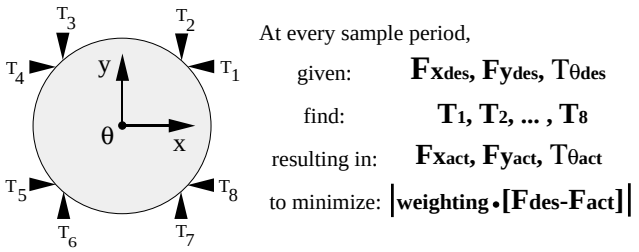

Figure 2: **Thruster Mapping Problem Definition**

The on-off thrusters and coupling between forces and torque make this problem difficult.

The base control strategy developed for this system is shown in Figure 3. Note that partitioning the controller into a "control module" and a "thruster mapper" greatly simplifies controller design since both components can be designed independently. Smooth actuation is still possible due to the low thruster impulse, which results from high sample rate (60 Hz.), low thrust (force per thruster, $F = 1$ N; torque per thruster, $\tau = 0.14$ N-m) and high mass (mass, $M = 70$ kg; moment of inertia, $I = 3.1$ kg-m^2).

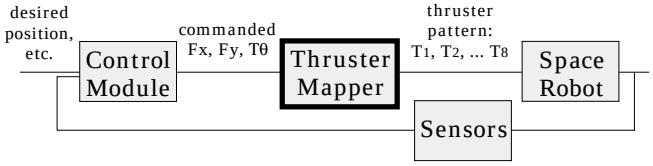

Figure 3: **Base Control Strategy**

The control module treats the thrusters as linear actuators. The thruster mapper must find the thruster pattern producing a force closest to that requested by the base control module.

The thruster mapping task, shown in Figure 2, that must be performed during each sample period is to take an input vector of continuous-valued desired forces and torques, $[F_{xdes}, F_{ydes}, \tau_{\theta des}]$, and find the output vector of discrete-valued (off, on) thruster values, $[T_1, T_2, ..., T_8]$, that minimizes a specified cost function.

In this application, the cost function was chosen to be the length of the normalized force error vector, where the normalization factors were the force-per-thruster, $F_{thruster}$, and torque-per-thruster, $T_{thruster}$. The resulting cost function is $[1/F_{thruster}\ \ 1/F_{thruster}\ \ 1/T_{thruster}]*(F_{des}-F_{act})$. Other cost functions are possible. For example, one could involve weightings based on acceleration error (scale the above by the vehicle mass properties), or gas usage. If gas usage were used, the thruster mapper would trade off force error for a reduction in gas usage, just as an optimal controller balances error with control effort.

If the thrusters are all the same strength (the nominal configuration assumed in this example), firing two opposing thrusters (e.g. 1 and 4) will produce no net thrust. To eliminate these useless combinations, the eight on-off thrusters, $[T_1, T_2, ..., T_8]$, may be considered as four backwards-off-forwards thrusters $[R_1, R_2, R_3, R_4]$, where, for example, R_1 represents the reaction force resulting from T_1 and T_4. This reaction force representation is used here to reduce complexity.

The focus of this example is developing a neural network to implement the "Thruster Mapper" component. A subsequent step, made possible by the developments in Section 4, is to merge the base controller and thruster mapper design into a single component. This could potentially result in improved total system performance.

2.2 Solution Methods

Three different techniques have been applied to solve the thruster mapping problem in order to make evaluation of performance and comparisons possible. The first method is a non-neural network approach that provides the true optimum. The second uses a NN with direct training. The third uses a NN with indirect training.

These three solution techniques are summarized in Figure 4. The first implementation, *SEARCH*, used an exhaustive search at each sample period to find the thruster pattern minimizing the force error vector [2]. Symmetries are used to reduce the search space, but this method relies on testing every possible thruster pattern at each sample period to find the one with minimum error.

In the second method, *DIRECT TRAINING*, a neural network was trained to emulate the optimal mapping produced by the exhaustive search [3]. The reason this *DIRECT TRAINING* approach was investigated was that it allows the study of network architecture and topology issues before tackling the additional problems that come with indirect learning. Hence it serves as a stepping stone to the goal of indirect learning. The approach also has potential advantages. In particular, using a NN as a function emulator may increase computational speed and system robustness due to the distributed, parallel nature of the computation.

The investigation of the network topology issues associated with this *DIRECT TRAINING* approach led to

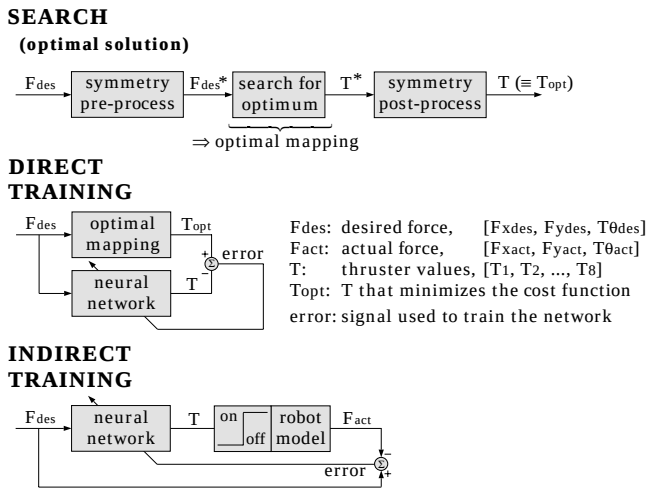

Figure 4: **Thruster Mapping Methods**

the Fully Connected Architecture, presented in Section 3. The FCA can also be used with the indirect training method described below.

In the third method, *INDIRECT TRAINING*, a neural network was trained to find the optimal solution when presented with a model of the plant, but no optimal teacher. This required back-propagation of error through the discrete-valued thrusters, which motivated development of the noise injection method presented in Section 4. The benefit of this method over *SEARCH* and *DIRECT TRAINING* is that it provides a framework for adaptation, and eliminates the need to solve the problem manually (find the optimal solution through analysis).

When evaluating mapping performance, the search method represents a lower bound, since it defines the optimal solution. Direct-training performance will be used as a benchmark for comparison with indirect training, since it represents the lower bound defined by the finite mapping complexity available with the chosen network topology.

3 Fully-Connected Architecture

A number of issues are present in the thruster mapping control application discussed above that are common to many NN control problems.

- prior information about the system exists, and it should be possible to exploit this information when generating the NN
- initial learning speed is important if the NN will be trained on-line
- the NN topology required to achieve an accurate mapping without over-fitting is unknown
- some of the control outputs (thruster values) influence one another (e.g. directly opposing thrusters should never fire together).

In this section, a general network architecture that addresses these issues is suggested. This "Fully-Connected Architecture" is for feed-forward neural networks that can be trained using backpropagation [4] [5], and refers to the structure shown in Figure 5. It was first presented by Werbos [6], and initially developed in a control context by Wilson and Rock [3]. The network's neurons are considered to be ordered, beginning with the first input, ending with the last output, and having hidden units in between, perhaps interspersed among input or output units. Note that there is no longer a concept of layers. Backpropagation restricts information flow to one direction only, so to get maximum interconnections, each neuron takes inputs from all lower-numbered neurons and sends outputs to all higher-numbered neurons.

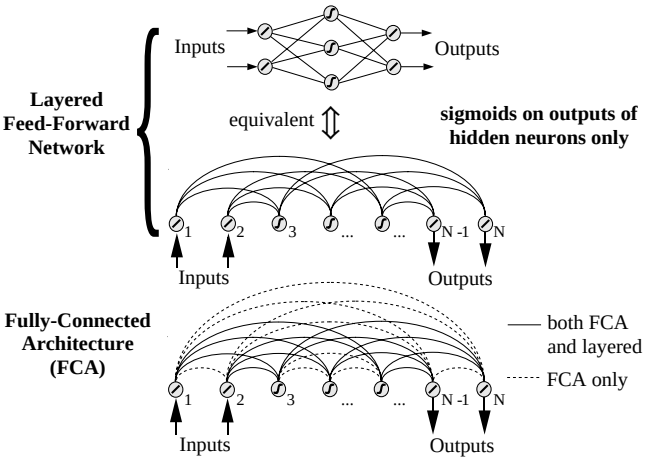

Figure 5: **Extra connections available with FCA**

This general feed-forward architecture subsumes more-familiar single or double-hidden-layer architectures. Here, the FCA is shown to have all the connections of a single-hidden-layer network, and some extras as well.

3.1 Comparison with layered network

Figure 6 highlights the benefits of the extra connections that are unused in a single-layered network. In particular:

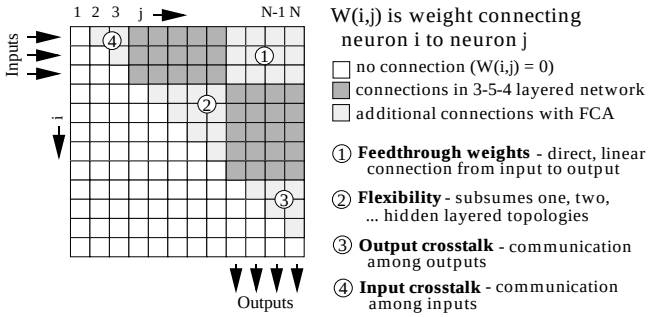

Figure 6: **Weight matrix representation to highlight benefits of Fully-Connected Architecture**

- Feedthrough Weights: this segment, shown in region 1 in Figure 6, is a matrix that implements a direct, linear connection from inputs to outputs (provided sigmoids are used only on hidden units). This provides fast initial learning and allows *direct preprogramming* of a linear solution calculated by some other method. This is particularly important for control applications, where there is a large body of linear control knowledge that can be drawn upon to provide a good starting point. The FCA provides for seamless integration of linear and non-linear components
- Flexibility: since the FCA subsumes any number of hidden layers, when combined with a systematic weight pruning procedure, the network topology (defined by the remaining connections) is set in a systematic manner based on gradient descent. The weights shown in region 2 of Figure 6 represent the flexibility of the FCA in that the connections may be configured to provide one and two hidden layer topologies (in general, any feed forward network topology).
- Cross-talk among inputs and outputs: these connections, shown in regions 3 and 4 of Figure 6 may be valuable, i.e. one output may excite or inhibit another output, a feature unavailable with layered networks.

The "disadvantages" (i.e. issues which must be addressed) of the FCA include:

- Increased complexity: number of weights increases quadratically with the number of hidden units, versus linearly for a layered architecture. The extra weights increase susceptibility to over-fitting.
- Slower hardware implementation: updating must be one neuron at a time, versus one layer at a time for layered networks.

The question is whether the benefits of the enhanced functionality outweigh the increased computational load and susceptibility to over-fitting. This must be decided for each application.

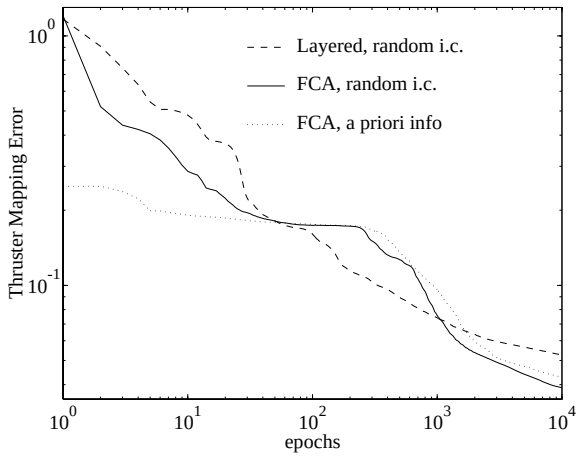

Figure 7: **Training History, FCA versus Layered Networks**

Figure 7 compares learning histories (thruster mapping error on the training set) for the thruster mapping problem (with direct training) outlined in the previous section. The networks, each with 5 hidden neurons, were trained to emulate the optimal mapping (minimizing force error). Each curve in the figure represents the average performance for ten different sets of initial weights.

Looking at the initial learning performance, the FCA network performs better than the layered network, due to the weight gradient being instantly available *via* the direct connection of inputs to outputs. As expected, the FCA network with the *a priori* linear solution built in provides the best early performance.

The *a priori* linear solution used here was found by assuming that the thrusters are capable of continuous-valued thrust output (a linearized version of this problem). The solution is simply a 4x3 pseudo-inverse of the 3x4 matrix which maps reaction forces, R, to base forces, F. Recognizing that the direct feed-through segment of the fully-connected network provides exactly this computation (output = weight matrix $\times$ input), it is possible to incorporate this *a priori* knowledge by putting the pseudo-inverse linear solution directly into that submatrix, as an initial condition for the weight matrix.

In the middle region, between 100 and 1000 epochs, the layered network performance surpasses that of the FCA, due to the reduced number of parameters, and simplified search space. However, after 1000 epochs, the greater functionality of the FCA network comes into play and performance surpasses that of the layered network.

3.2 Complexity control

The above has shown the potential value of the extra connections associated with a fully-connected neural network, both in faster initial learning, and in better final performance. However, the high number of parameters, while increasing functionality, makes the network susceptible to over-fitting. Often during training, performance on test and training sets will improve until a certain point, and then test performance will worsen as the network stops generalizing, and begins to fit the particular data set, as seen in Figure 8. Use of a "sufficiently-large" training set can reduce over-fitting problems, but this may not be practical due to a lack of data, or an adaptation speed requirement that requires a faster solution than this data-intensive brute-force approach.

Many systematic network pruning techniques have been proposed. One used successfully in this work involves the addition of a complexity cost term to the total cost function, as first proposed by Weigend and Rumelhart in [7]. Each weight contributes $\lambda \cdot (w_s^2/(w_s^2+1))$ to the total cost function, where $w_s = w/w_0$ is a scaled value of the weight. The scale factor, w_0, effectively sets the cutoff point for weights, and λ selects the relative importance of complexity cost versus performance cost.

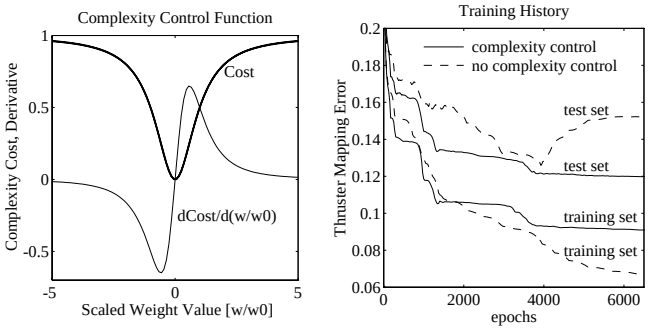

Figure 8: **Complexity Control Function, Effect on Network Performance**

The complexity control function and training histories for a fully-connected network with 5 hidden neurons are plotted in Figure 8. Without complexity control, over-fitting becomes clear at around the 4000th epoch, as the performance on the test set worsens, while performance on the training set improves. With the addition of the complexity term, over-fitting is controlled, as performance histories on test and training sets no longer diverge.

4 Backpropagation Learning for Discrete-Valued Functions

The previous section dealt with direct training, and led to the development of the network architecture used to implement the thruster mapping. To allow indirect training, where the learning signal (error) is generated based on the robot model output (rather than an optimal teacher), the error must be backpropagated though the robot model. The discontinuity introduced by the use of on-off thrusters on the robot presents a significant obstacle, and necessitates the development of the training method presented here.

For direct training, the problem of discrete-valued functions does not arise because the discrete values are supplied as the output patterns in the training set (e.g. [1.87 -0.76 0.11] gets mapped to [0 0 1 1 1 0 0 0]), rather than as a discontinuous function.

4.1 Problem Statement

Optimization methods that use gradient information often converge much faster than those that do not. Use of the backpropagation algorithm to get this gradient information for training neural networks has made them useful in many applications; however, BP's requirement of continuous differentiability, not only for the network itself, but for anything that the error is backpropagated through (e.g. the plant model in a control problem), limits its applicability.

This is a significant limitation since there are many applications where discrete-valued states arise. For example: on-off thrusters commonly used in spacecraft (the example used in this paper); other systems with discrete-valued inputs and outputs; and NNs built with signums (also known as hard-limiters or Heaviside step functions) rather than sigmoids. Signum networks may be preferred to sigmoidal ones due to hardware considerations.

In cases like these, one choice is to use an alternative method not restricted to continuously differentiable functions, such as unsupervised learning, simulated annealing, or a genetic algorithm, but these are usually significantly slower to train, because they do not use gradient information.

Another option is to approximate the discrete-valued functions with linear functions or smooth sigmoids during the learning phase, and switch to the true hard-limiting functions at run-time, as with the original ADALINE [8]. This method works in many cases where the behavior of the system with sigmoids is close enough to that of the real system. However, this assumption is un-

reliable, and the thruster control problem presented here offers a clear example where this method fails.

In related research aimed at using gradient-based learning for multi-layer signum networks, Bartlett and Downs [9] use weights that are random variables, and develop a training algorithm based on the fact that the resulting probability distribution is continuously differentiable. The algorithm is limited to one hidden layer, requires all inputs to be 1 or -1, and needs extra computation to estimate the gradient.

In this section, a technique for BP learning for systems with discrete-valued functions is presented and applied to the on-off thruster control problem described in Section 2. Further results, including application to training of multi-layer signum networks, are presented in [10].

4.2 Noisy sigmoid training algorithm

The basis of the method is to replace the discrete-valued functions with approximating functions composed of "noisy sigmoids". Application of the method to the training of a single hard-limiting neuron is shown in Figure 9. The first block diagram shows the neuron as it appears at run-time: a dot-product and hard-limiter. The next two diagrams show the neuron during training, where the hard-limit from the top diagram has been replaced by a smooth sigmoid function.

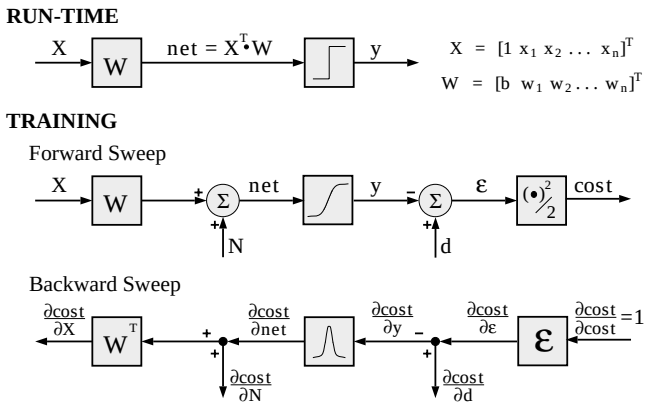

Figure 9: **Training Algorithm**

During training, replace discrete-valued signums with sigmoids, and inject noise before the sigmoid on the forward sweep. The backward sweep calculation is the same as standard backpropagation.

This is almost the same as training a standard neuron with backpropagation – the only difference involves the injection of zero-mean noise, N, immediately before the sigmoid. The noise injection enters just as the desired signal does, and does not corrupt the calculation of $\partial \text{cost}/\partial W$, the gradient estimate used in the weight update rule. Using an unmodified backward sweep is not only the simplest thing to do, it does precisely the right calculations for estimating the weight gradient.
To summarize, the training algorithm becomes:

- Replace the hard-limiters with sigmoids during training
- Inject noise immediately before the sigmoids on the forward sweep
- Use the exact same backward sweep as with standard backpropagation

4.3 Intuitive explanation

Without addition of noise, the network may train using sigmoid output values in the sigmoid transition region (roughly -0.8 to 0.8) that will be unavailable at run-time. Simply rounding off at run-time may introduce significant errors. For example, in a hypothetical cost surface, a value of 0.4 may be optimal, but if forced to choose between -1 and 1, a value of -1 may be better.

The goal of noise injection is to move neuron activations away from the transition region, so round-off error will be small when the discrete-valued functions are replaced. For this reason, the standard deviation of the noise is chosen to be higher than the width of the transition region of the sigmoid.

Figure 10 shows how the neuron output distribution changes as the noise level increases. With no noise, only a single output can result, but as noise increases to cover most of the transition region, the output distribution approaches that of a hard-limiting function. Differentiability is maintained, however, so gradient information will be available to speed up learning. Since the noise has pushed the distribution to approximate a hard-limiting non-linearity, when the hard-limiter is re-introduced at run-time the performance degradation will be small.

4.4 Extensions, application considerations

This method has been successfully applied to multiple layers of hard-limiting units with no further modification [10]. One concern is the attenuating effect of the derivative-of-sigmoid function. When back-propagated through many layers of near-saturated sigmoids, the error signal is attenuated and may lead to slow learning. To handle this problem, it may be necessary to increase gradually the noise variance - slowly pushing the outputs from the linear region to the hard-limits, rather than all

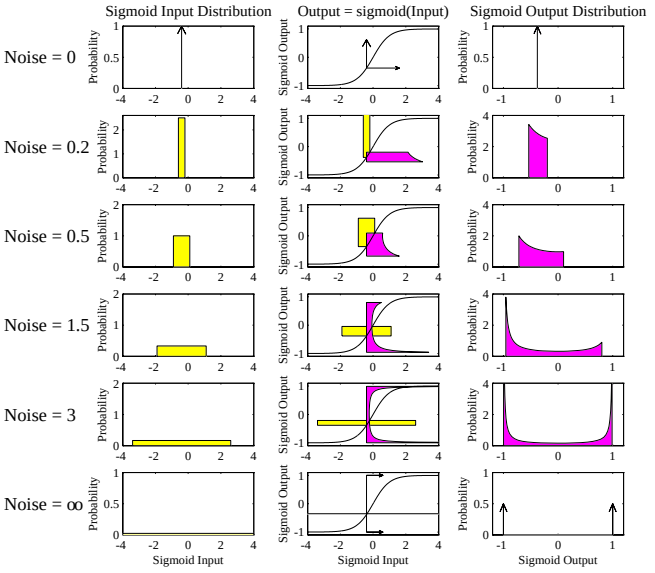

Figure 10: **Effect of Input Noise Level on Sigmoid Output Distribution**

Lightly-shaded region in column 1 represents the sigmoid input probability distribution (in this case, $-0.3+$ uniformly distributed noise). Darkly-shaded region in column 3 is the sigmoid output distribution (from -1 to 1). Each distribution has an area of 1. Input and Output are plotted together in column 2 to show how the sigmoid produces this input-output relationship. As noise level increases, and the input distribution spreads out, the sigmoid output approaches that of a hard-limiter, while remaining differentiable.

at once, where the attenuation is high and the network will find it difficult to react.

When using a system with discrete-valued functions that are not Heaviside step functions, the method may work if a continuously differentiable approximating function is used. For example, a function whose output can take on a number of discrete values may be approximated by combining a number of sharp sigmoid functions. This was done for the thruster mapping, replacing the eight (0,1) thrusters with the equivalent set of four (-1,0,1) thrusters.

4.5 Application to space robot

In order to demonstrate this new training procedure, it was applied to the thruster mapping with indirect training, as shown in the third section of Figure 4. In this case, the optimal mapping is not used, and the NN must learn the mapping through experimentation with the plant model.

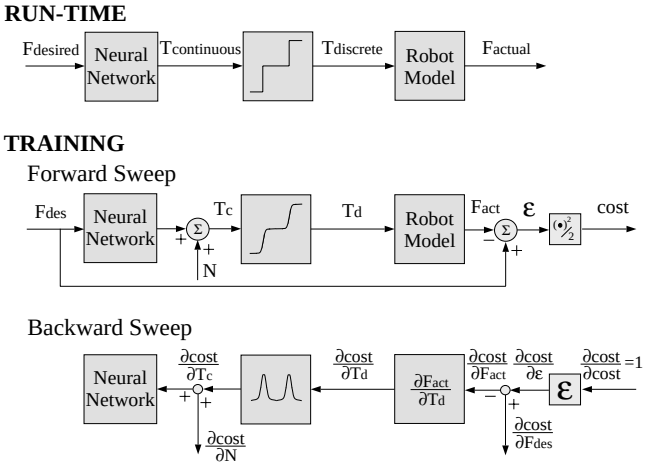

Figure 11: **Thruster mapping, indirect training method**

Training without this noise-injection technique produces large errors because the discrete-valued nature of the thrusters is not enforced during network training, and large roundoff errors result at run-time. For example if one unit of thrust is requested in the $+x$ direction, during training, the network will set T_4 and T_5 to $+0.5$, but at run time, for requested forces near 1.0, T_4 and T_5 are likely to both be 0 or both be 1, resulting in a large error.

Figure 12 shows the result of indirect training with two differentiable thruster models. During training with the continuous thruster models, the NN produces a mapping with a very low error, which is not plotted here. However, when the signums are replaced at run-time, the error is large, and is the "thruster mapping error" plotted in the bottom of Figures 12 and 13. The errors are high because the network learned to optimize the solution using outputs that would be unavailable at run-time. The resulting roundoff error is unknown to the NN during training.

In Figures 12 and 13, each dot represents the final performance after a 10,000 epoch training run. The shaded regions represent $mean \pm \sigma$ performance for ten runs.

Figure 13 shows the results when the thrusters are modelled by *noisy* tri-level sigmoids. With $noise = 0$, error is high, corresponding to the data in Figure 12, but as noise increases, performance approaches that of the network trained directly (emulating the optimal mapping).

The direct-training performance represents a lower bound set by the functional complexity of the $3-10-4$ layered network. The best noise value in this application

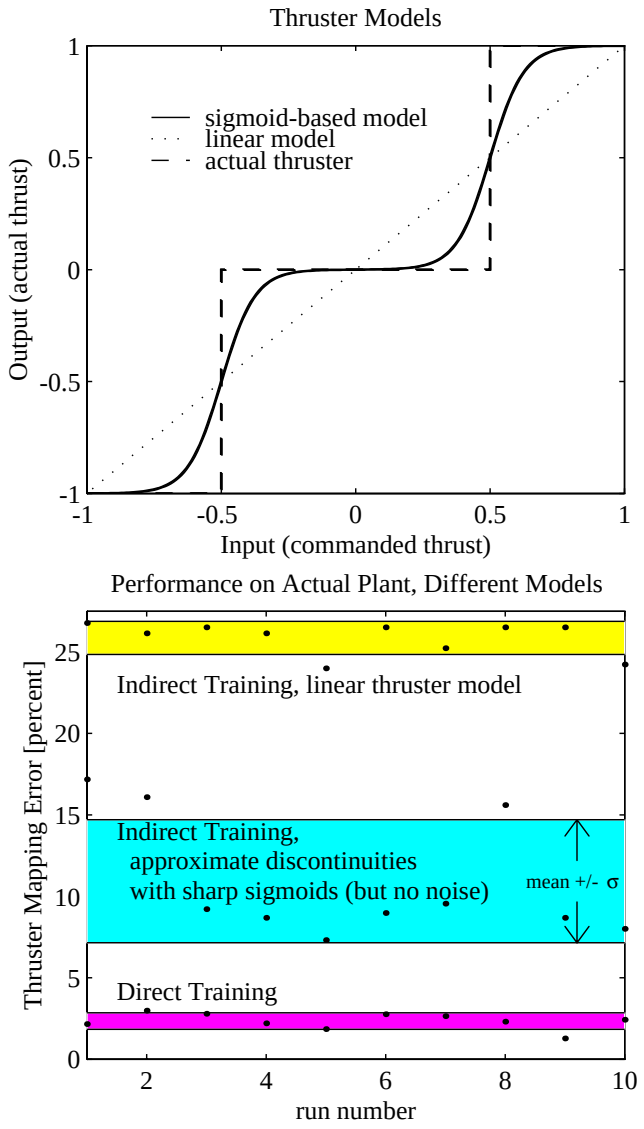

Figure 12: **Results of indirect training, two differentiable thruster models**

The sigmoid-based approximation (without noise) is better than the linear model, but has limited performance. The results from direct training represent a lower limit for comparison. Mapping error is average percent error above the optimal mapping (which results from an exhaustive search of all possible thruster combinations). The shaded areas represent the mean ± σ for ten different runs. 3 − 10 − 4 layered networks were used.

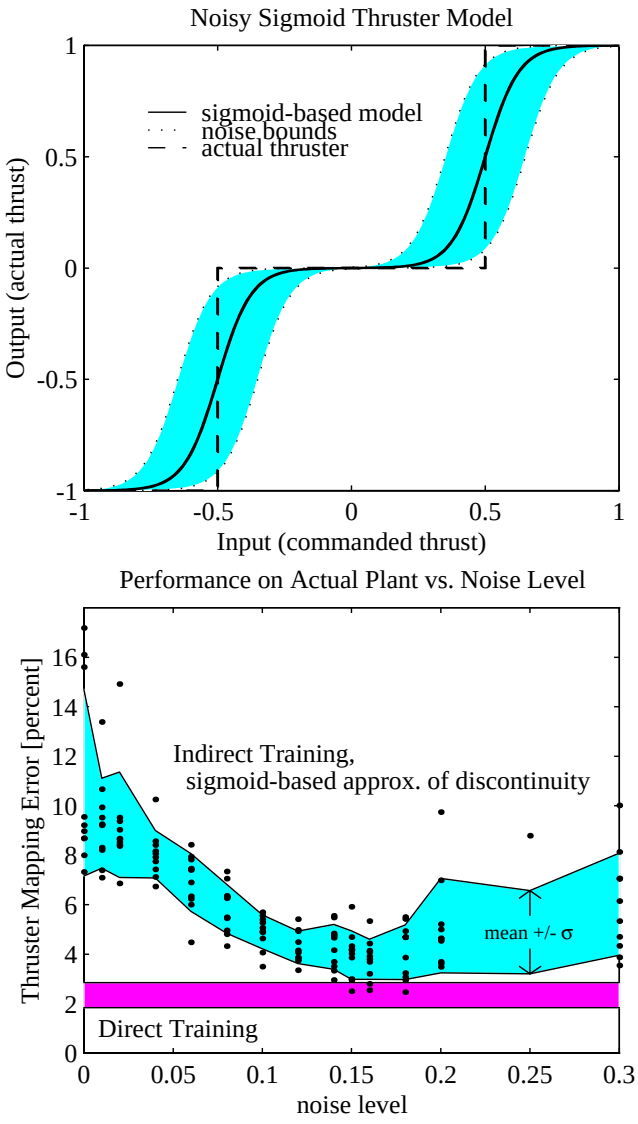

Figure 13: **Results of indirect training, noisy tri-level sigmoid thruster model**

Top: the sigmoid sharpness factor (slope at the midpoint = 4) and noise level (0.15) for the noisy tri-level sigmoid appear to be intuitively correct. Bottom: as noise increases, performance approaches that of the network trained directly (emulating the optimal mapping). 3 − 10 − 4 layered networks were used.

seems to be around 0.15, and the resulting noisy sigmoid is shown in the top of Figure 13. Examining this figure, the sigmoid sharpness and noise levels seem to be set correctly according to intuition. As noise increases beyond 0.2, error increases as expected (the "off" region of the sigmoid becomes blurred). The method is fairly robust to the noise value selected, and the effect of noise level on performance makes intuitive sense.

A good solution results when noise is added because it prevents the network from using a solution that uses non-saturated portions of the tri-level sigmoid. Such a solution would give a nearly random output and high error during training. The training algorithm must find a solution that works well *despite* the noise addition. This

means the expected value of the output must be well into the saturated regions to work consistently well. The resulting mapping approximates the optimal solution very well.

5 Experimental Results

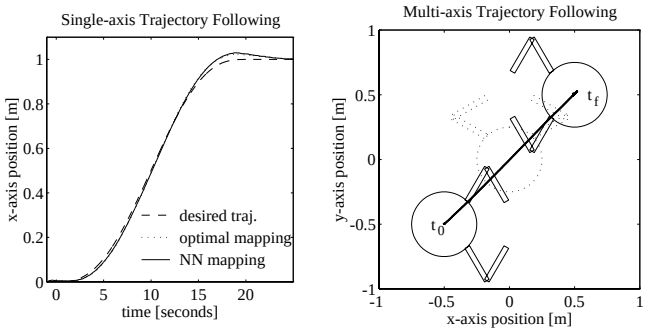

Figure 14: **Trajectory-Following Performance**

Experiments were performed on the mobile robot described in Section 2 to verify the applicability of these neural network results. The FCA neural network thruster mapping component described in Section 3 is implemented on the on-board Motorola® 68040 processor, as is the rest of the control system (at a 60 Hz. sample rate).

Figure 14 shows robot base position during single-axis and multi-axis maneuvers. For the single-axis maneuver (left), good tracking is obtained from both optimal and neural network thruster mapping components. In the 3-axis maneuver (right), the neural network control system closely follows the 20-second-long straight-line trajectory in (x, y, θ). Tracking error is very small, so to avoid clutter, only the robot's actual (x, y) position is plotted.

If a layered network had been used in place of the FCA, similar tracking performance could have been achieved with an increased number of hidden neurons, and direct training. However, if training indirectly without the noisy sigmoid learning method, the high mapping errors seen in Figure 12 would appear and result in large tracking errors.

6 Summary and Conclusions

This paper has described two recent developments in neural network control that grew out of a research program using a laboratory-based prototype of a free-flying space robot. Both advances were motivated by, and developed for, a complex thruster mapping function typical of real spacecraft.

A fully-connected neural network architecture was presented that has connections beyond those provided by a layered network, yet is trainable with backpropagation. Aided by a systematic complexity control scheme, this network was shown to have certain advantages over layered networks, particularly for control problems.

A new technique was developed that extends BP learning to systems involving discrete-valued functions (which are not continuously differentiable), such as the on-off thrusters used to control our robot. The modification to BP is very small, simply requiring careful injection of noise on the forward sweep, yet the improvement in network performance is dramatic.

When tested experimentally on the real robot, all networks provided near-optimal performance during multi-axis trajectories, thus demonstrating the utility of these techniques.

References

[1] W. Thomas Miller III, Richard S. Sutton, and Paul J. Werbos, editors. *Neural Networks for Control.* Neural Network Modeling and Connectionism. The MIT Press, Cambridge, MA 02142, 1990.

[2] Marc A. Ullman. *Experiments in Autonomous Navigation and Control of Multi-Manipulator, Free-Flying Space Robots.* PhD thesis, Stanford University, Stanford, CA 94305, March 1993.

[3] Edward Wilson and Stephen M. Rock. Experiments in control of a free-flying space robot using fully-connected neural networks. In *Proceedings of the World Congress on Neural Networks*, volume 3, pages 157–162, Portland OR, July 1993.

[4] Paul J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University, Cambridge, MA 02142, August 1974.

[5] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In David E. Rumelhart, James L. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing*, page 318. The MIT Press, Cambridge, MA 02142, 1986.

[6] Paul J. Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, October 1990.

[7] Andreas S. Weigend, Bernardo A. Huberman, and David E. Rumelhart. Predicting the future: A connectionist approach. *International Journal of Neural Systems*, 1(3):193–209, 1990.

[8] B. Widrow and R. Winter. Neural nets for adaptive filtering and adaptive pattern recognition. *IEEE Computer*, 21(3):25–39, March 1988.

[9] P.L. Bartlett and T. Downs. Using random weights to train multilayer networks of hard-limiting units. *IEEE Transactions on Neural Networks*, 3(2):202–210, March 1992.

[10] Edward Wilson. Backpropagation learning for systems with discrete-valued functions. In *Proceedings of the World Congress on Neural Networks*, San Diego CA, June 1994.