EXPERIMENTS IN NEURAL NETWORK CONTROL OF A FREE-FLYING SPACE ROBOT

Edward Wilson*
Aerospace Robotics Laboratory
Stanford University
Stanford, California 94305
ed@sun-valley.Stanford.edu

Abstract

Recent developments in neural network control at the Stanford Aerospace Robotics Laboratory are presented. A "Fully-Connected Architecture" (FCA) is developed for use with backpropagation (BP). This FCA has functionality beyond that of a layered network, and these benefits are shown to be particularly beneficial for control tasks. A complexity control method is successfully used to manage the extra connections provided, and prevent over-fitting. A technique that extends BP learning to discontinuous functions is presented and applied to a difficult on-off thruster control problem. This method is also applicable to networks of hard-limiting neurons. The modification to BP is very small, simply requiring careful 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.

1 Introduction

This paper outlines research on neural network control at the Stanford Aerospace Robotics Laboratory. In our research program, we have developed some widely-applicable neural network methods from our efforts to apply networks to the control of a free-flying space robot and flexible structures.

In Section 2, the robot is described, and the particular thruster mapping problem we address is presented. Controlling the on-off thrusters presents a truly challenging problem for the neural network application. Work on this problem has led to the contributions described in Sections 3 and 4. Section 3 presents a "Fully-Connected Architecture" (FCA) for use with backpropagation (BP), that has greater functionality than a standard layered network. Particular benefits of the FCA, some of which are especially useful for control problems, are outlined. Section 4 presents a method for using BP learning with systems containing discontinuous functions, such as the on-off thrusters on our robot. The method is a simple modification to standard BP, and extends to multiple layers of hard-limiting neurons or the FCA without modification. The experimental viability of these methods was verified on our robot, and is presented in Section 5.

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

2 Robot Control Application

The control task that motivated the neural network developments was 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.

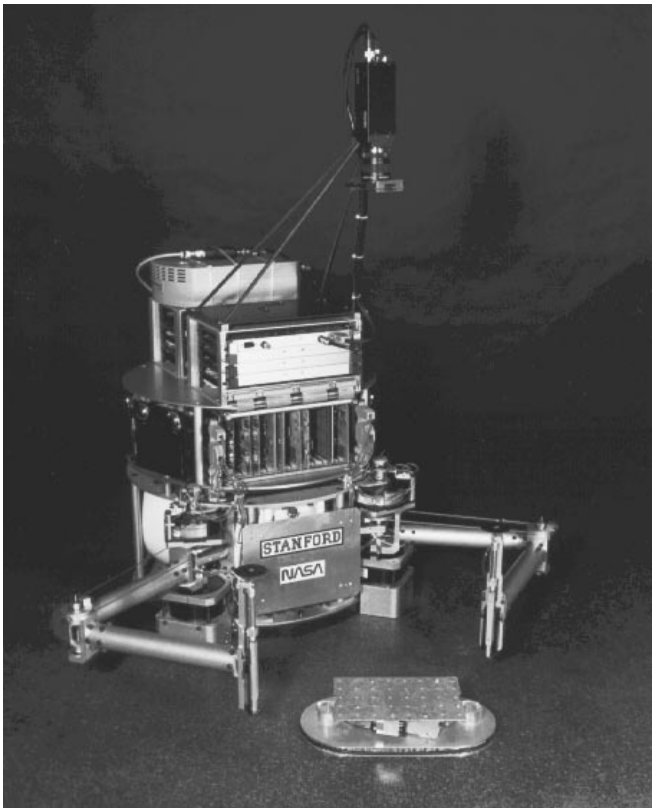

Figure 1: **Stanford Free-Flying Space Robot**

Experiments are performed using a mobile robot that operates in a horizontal plane, using air-cushion technology 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 a 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 [Ullman 93].

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

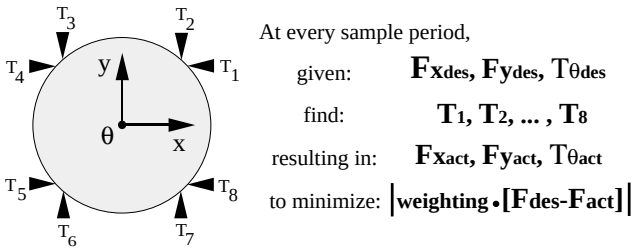

Figure 2: **Thruster Mapping Problem Definition**

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

The on-off thrusters substantially complicate the control design, due to their discontinuous nature and the fact that each thruster simultaneously produces both a net force and torque. The current base control strategy is shown in Figure 3. In this paper, we focus on developing neural networks for 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, improving performance.

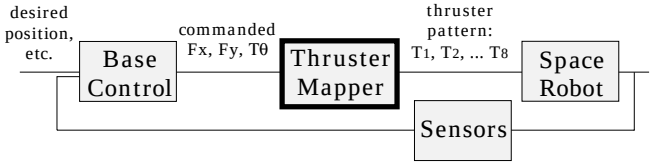

Figure 3: **Base Control Strategy**

Three different techniques used to solve the thruster mapping problem are summarized in Figure 4. The first implementation used an exhaustive search at each sample period to find the thruster pattern minimizing the force error vector [Ullman 93]. Symmetries are used to reduce the search space, but this method relies on testing every possible thruster pattern to find the one with minimum error.

The second method used a neural network that had been trained off-line to emulate the optimal mapping produced by the exhaustive search [Wilson and Rock 93]. This did not *require* any new neural network developments, but led to the Fully Connected Architecture, presented in Section 3.

The third method used a neural network trained to find the optimal solution when presented with a model of the plant, but no optimal teacher. This required backpropagation of error through the discontinuous thrusters,

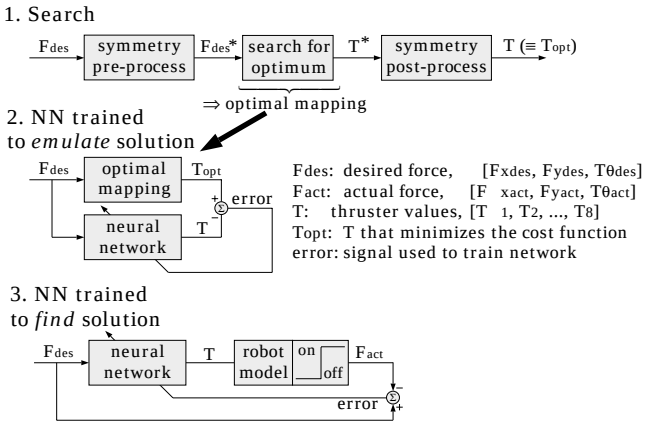

Figure 4: **Thruster Mapping Methods**

which was made possible by the noise injection method presented in Section 4.

3 Fully-Connected Architecture

3.1 Introduction

We present a general architecture for feed-forward neural networks that can be trained using backpropagation [Rumelhart *et al.* 86] [Werbos 74]. This "Fully-Connected Architecture" refers to the structure shown in Figure 5, and first mentioned in [Werbos 90] and analyzed in [Wilson and Rock 93]. Instead of layers, the network has neurons that are ordered, beginning with the first input, ending with the last output, and having hidden units in between, perhaps interspersed among input or output units. 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.

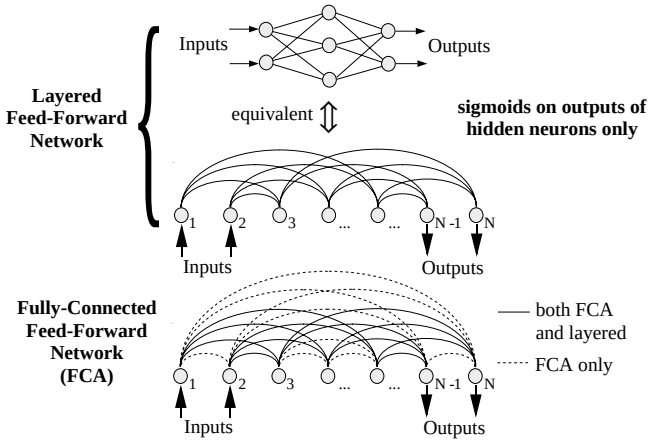

Figure 5: **Extra connections available with FCA**

This general feed-forward architecture subsumes more-familiar single or double-hidden-layer architectures. Here we show that it has all the connections of a single-hidden-layer network, and some extras as well.

3.2 Comparison with layered network

Figure 6 highlights the benefits of the extra connections that are unused in a single-layered network. The question is whether the enhanced functionality outweighs the increased computational load and susceptibility to over-fitting. This must be decided for each application.

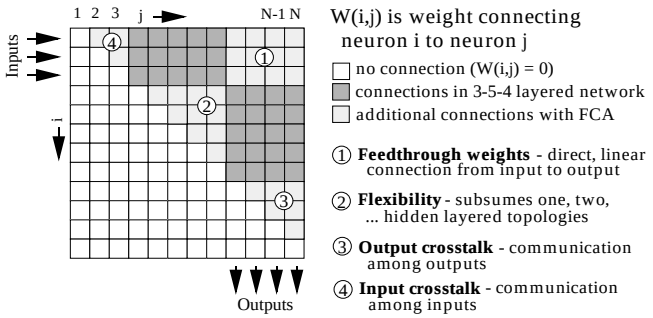

Figure 6: **Weight matrix representation to highlight benefits of Fully-Connected Architecture**

Advantages of the FCA:

- The Feed-through section 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 pre-programming* 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. This 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.
- Cross-talk among inputs and outputs may be valuable, i.e. one output may excite or inhibit another output, a feature unavailable with layered networks.

Disadvantages:

- 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.

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

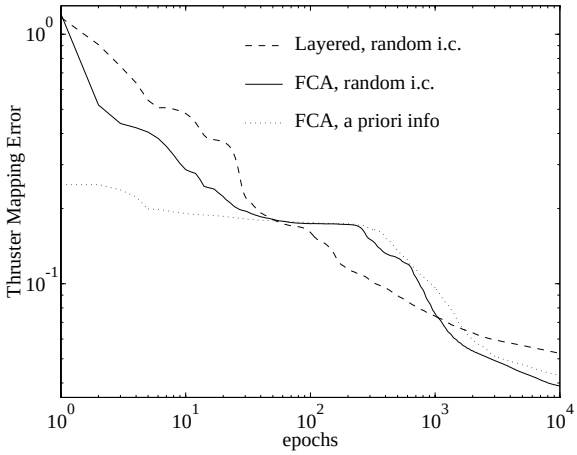

Figure 7: **Training History, FCA versus Layered**

Looking at the initial learning performance, the FCA network performs better, 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.

In the middle region, between 100 and 1000 epochs, the layered 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.

3.3 Complexity control

This experiment has shown the potential value of the extra connections associated with a fully-connected neural network. 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 brute-force approach.

Many systematic network pruning techniques have been proposed. One we have used successfully involves the addition of a complexity cost term to the total cost function, as first proposed in [Weigend *et al.* 90]. 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.

The complexity control function and training histories for a fully-connected network with 5 hidden neurons are plot-

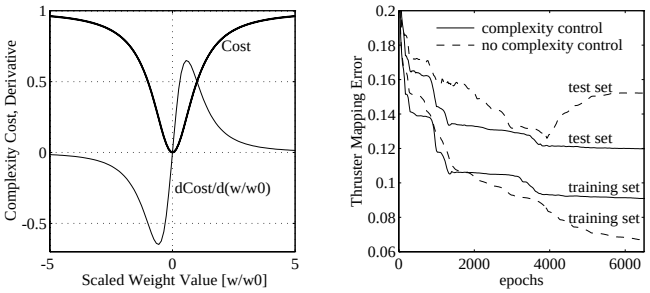

Figure 8: **Complexity Control Function, Effect on Network Performance**

ted 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 for Discontinuous Functions

4.1 Introduction

Optimization methods using gradient information often converge much faster than those that do not. The use of the backpropagation algorithm to get this gradient information for training neural networks has made them practical in many cases; however, BP's requirement of 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; other systems with discrete inputs and outputs; and networks built with step functions rather than sigmoids (such 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 by the absence of gradient information, such as unsupervised learning, simulated annealing, or genetic algorithm, but these are usually significantly slower to train, because they are not gradient-based.

Another choice is to approximate the discrete-valued functions with linear functions or smooth sigmoids during the learning phase, and switch to the true discontinuous functions at run-time [Widrow and Winter 88]. 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 unreliable, and the thruster control problem presented here offers a clear example where this method fails.

In related research, work has been performed on extending the BP algorithm to handle networks of neurons with hard-limiting output functions [Bartlett and Downs 92]. Bartlett and Downs 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 that all inputs be 1 or -1, and requires significant extra computation to estimate the gradient.

In this section we present a technique for BP learning for systems with discontinuous functions, and apply it to the on-off thruster control problem described in Section 2.

4.2 Training algorithm

We introduce the method of noise injection by applying it to training of a single hard-limiting neuron, as shown in Figure 9. This is equivalent to Rosenblatt's perceptron [Rosenblatt 62]. Our method is not especially useful here, but as will be shown, this method extends to multiple layers whereas the perceptron learning law does not.

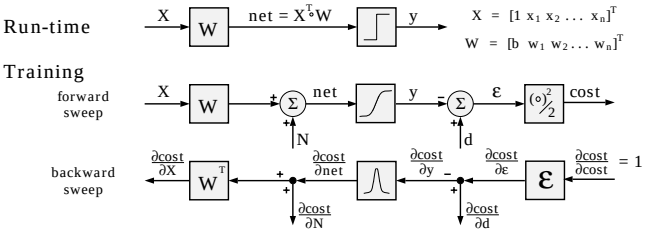

Figure 9: **Training Algorithm**

Replace discontinuous hard-limiters with sigmoids during training, and inject noise before the sigmoid on the forward sweep. The backward sweep calculation is the same as standard backpropagation.

The first block diagram in Figure 9 shows the neuron as it appears at run-time, a dot-product and hard-limiter. For simplicity in bookkeeping, we have augmented the input, X, and weight, W, vectors to include the threshold bias for the output function.

The next two diagrams in Figure 9 show the neuron during training, where the hard-limit from the top diagram has been replaced by a smooth sigmoid function. The input, X, is propagated through the forward sweep, finally resulting in an error, ϵ, and a cost. The derivative of this cost is calculated and propagated though the backward sweep, resulting in a $\partial\text{cost}/\partial X$ to be propagated to more units upstream, and a $\partial\text{cost}/\partial\text{net}$ to be used in calculating $\partial\text{cost}/\partial W$, which is used in the gradient descent algorithm.

This is exactly the same as training a standard neuron with backpropagation. The difference involves the injection of noise, N, immediately before the sigmoid. As will be shown below, as the standard deviation of the noise is increased to be larger than the transition region of the sigmoid, the resulting gradient can be used to train for the hard-limiter. An important thing to notice is that the noise injection enters just as the desired signal does, and does not affect the calculation of $\partial\text{cost}/\partial W$. 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 is:

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

4.3 Why it works

Without addition of noise, the network would train using values in the sigmoid transition region (roughly -0.8 to 0.8) which are unavailable on the real system. This might lead to poor performance. For example, a value of 0.4 may be optimal, but if forced to choose between -1 and 1, a value of -1 may be better. By moving away from the transition region, the noise forces the network to choose only between the limiting values.

An intuitive reason for adding the noise is to throw the neuron off its transition region, and effectively force it to hard-limit at the high or low value. 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.

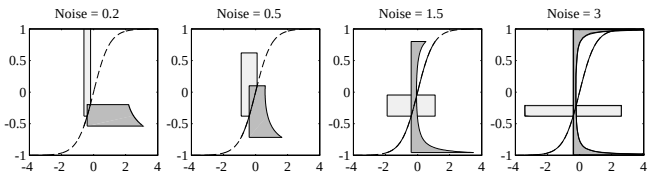

Figure 10: **Effect of Input Noise Level on Sigmoid Output Distribution**

Lightly-shaded region represents the sigmoid input distribution ($w \cdot x + noise$). Darkly-shaded region is the sigmoid output distribution, plotted horizontally to correspond to the sigmoid plot. As noise level increases, sigmoid output approaches hard-limiter output, while remaining differentiable.

4.4 Extensions, application considerations

This method extends to multiple layers of hard-limiting units with no modification. One concern that must be kept in mind while using it is the attenuating effect of the derivative-of-sigmoid function. When back-propagated through many layers of attenuation, the error signal becomes weak and can lead to slow learning. To handle this problem, it may be necessary to be gradual in increasing the noise variance - slowly push the outputs from the linear region to the hard-limits, rather than all at once, where the attenuation is high and the network will find it difficult to react.

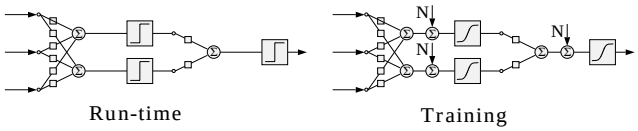

Figure 11: **Application to multi-layer network**

When using a system with non-differentiable functions that are not Heaviside step functions, the method will often work if a good 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.

The randomness introduced with the addition of noise may make learning slow because of the reduction in signal-to-noise ratio in the weight gradient estimation. We found that batch-learning using the exact same training set from one epoch to the next worked well (considering the "training set" to include the "input set" and "noise set"). Choosing a good training set, and fixing the noise results in a fixed deterministic cost hyper-surface. With a fixed cost function, adaptation of momentum and learning rate can be applied to allow quick optimization.

Application using pattern-learning should eventually converge to the same solution resulting from batch-learning. However, use of batch-learning facilitates on-line adaptation of the learning *parameters* (rate and momentum) which can dramatically improve convergence rate.

4.5 Application to space robot

Here, we apply this technique to the thruster mapping, as shown in the third section of Figure 4, where the network finds a solution itself, without the help of an optimal teacher.

Training without this noise-injection technique is not possible. 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, chances are they will both be 0 or both 1, resulting in a large error.

A good solution results when noise is added because it prevents the network from using a solution that uses non-saturated portions of the 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 region to consistently work well. The results approximate the optimal solution very well, and work when the sigmoids are replaced with signums.

The method worked well, and the thruster mapping problem was solved using both two-state (0,1) and three-state (-1,0,1) thrusters.

5 Experimental Results

Experiments were performed on the mobile robot described in Section 1 to verify the applicability of these neural network results. The neural network thruster mapping component is implemented on the on-board Motorola® 68040 processor, as is the rest of the control system (at a 60 Hz. sample rate).

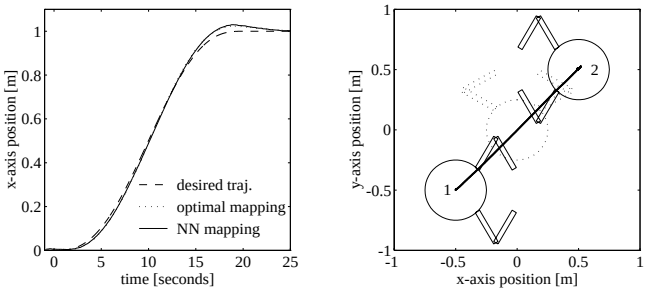

Figure 12: **Experimental Results, Single-Axis Maneuver, Multi-Axis Maneuver**

Figure 12 shows robot base position during single-axis and multi-axis maneuvers. A simple neural network control component designed using backpropagation through time [Werbos 90] [Nguyen and Widrow 90] is used in conjunction with the neural network component described in Section 3 of this paper. 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.

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 discontinuities, 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.

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

[Bartlett and Downs 92] 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.

[Nguyen and Widrow 90] D.H. Nguyen and B. Widrow. Neural networks for self-learning control systems. *IEEE Control Systems Magazine*, 10(3):18–23, April 1990.

[Rumelhart *et al.* 86] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. In David E. Rumelhart and James L. McClelland, editors, *Parallel Distributed Processing*, page 318. The MIT Press, Cambridge, MA 02142, 1986.

[Rosenblatt 62] F. Rosenblatt. *Principles of Neurodynamics.* Spartan Books, Washington, 1962.

[Ullman 93] M.A. Ullman. *Experiments in Autonomous Navigation and Control of Multi-Manipulator, Free-Flying Space Robots.* PhD thesis, Stanford University, Stanford, CA 94305, March 1993.

[Werbos 74] P.J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University, Cambridge, MA 02142, August 1974.

[Werbos 90] P.J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, October 1990.

[Weigend *et al.* 90] A.S. Weigend, B.A. Huberman, and D.E. Rumelhart. Predicting the future: A connectionist approach. *International Journal of Neural Systems*, 1(3):193–209, 1990.

[Wilson and Rock 93] E. Wilson and S. 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*, Portland OR, July 1993.

[Widrow and Winter 88] B. Widrow and R. Winter. Neural nets for adaptive filtering and adaptive pattern recognition. *IEEE Computer*, 21(3):25–39, March 1988.

Biography

Edward Wilson received B.S. degrees in Mechanical Engineering and Physics, and a M.S. degree in M.E. from the Massachusetts Institute of Technology, all in 1987. Master's thesis: "Design and Construction of a Laser Machine Tool for Processing Advanced Materials". Hughes Aircraft Company, 1987 to present. US Air Force Reserve officer in the Advanced Technology Branch / Microelectronics Section at McClellan AFB, CA, 1990 to present. Ph.D. candidate at Stanford University, Dept. of Mechanical Engineering, working with control applications of neural networks in the Aerospace Robotics Laboratory, 1988 to present. Current research interests include robotics, control, and neural networks. Tau Beta Pi, Sigma Xi, Pi Tau Sigma, ASME, IEEE.