Experiments in Control of a Free-Flying Space Robot Using Fully-Connected Neural Networks

Edward Wilson[*] Stephen M. Rock[†]
Stanford University
Aerospace Robotics Laboratory
Stanford, California 94305

Abstract

A general network architecture is presented that is capable of exploiting the full capabilities of the backpropagation algorithm. The extra connectivity of this architecture, which is unavailable in a layered network, allows seamless integration of linear *a priori* solutions, communication between output neurons, and greater overall functionality than a layered network. The increase in parameters can exacerbate over-fitting problems, and a systematic complexity control method is successfully demonstrated that lessens this problem. The viability of these methods is tested by applying them to a thruster mapping problem characteristic of space robots. Realizability is shown in an experimental demonstration on a 2-D laboratory model of a free-flying space robot.

1 Introduction

In the literature, the term "fully-connected feed-forward neural network", usually refers to a *layered* network, with an input layer, one or more hidden layers, and an output layer. "Feed-forward" indicates that signals flow from the input layer, through hidden layers, and to the output layer in one direction only, which is required by the backpropagation algorithm. "Fully-connected" indicates that every input is connected to every neuron in the first hidden layer, and so on between successive layers. While this layered architecture may be particularly well suited for many applications and certain hardware implementations, a more general structure may be able to take advantage of the full capabilities offered by the backpropagation algorithm [1].

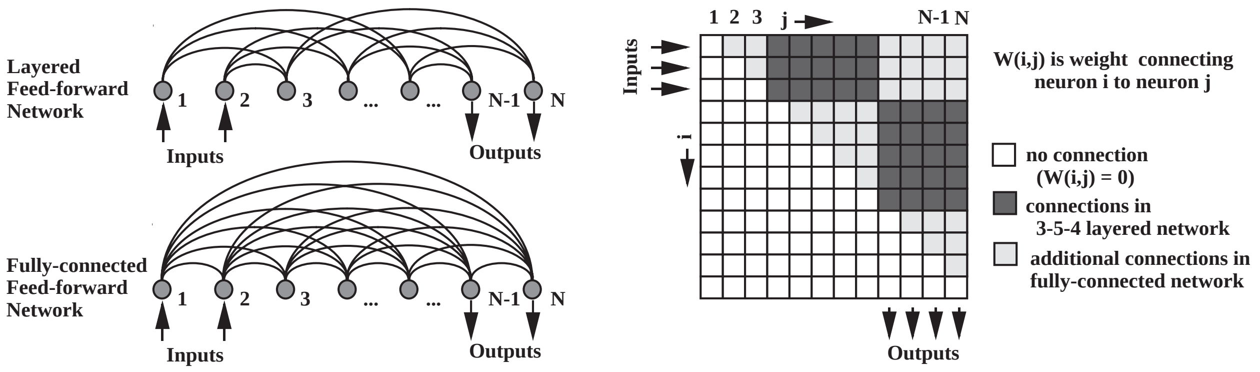

Layered Feed-forward Network
1
2
3
...
...
N-1
N
Inputs
Outputs
Fully-connected Feed-forward Network
1 2 3 j N-1 N
Inputs
i
Outputs
W(i,j) is weight connecting neuron i to neuron j
no connection (W(i,j) = 0)
connections in 3-5-4 layered network
additional connections in fully-connected network

Figure 1: **Extra connections available with fully-connected feed-forward networks**

In this paper, the term "fully-connected" will refer to the structure shown in Figure 1. Instead of layers, a fully-connected network can be considered to have 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 [2]. 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. For example, the last output neuron takes inputs from all the hidden neurons, just as in a layered architecture; however, it also takes inputs from each of the input neurons and previous output neurons.

[*]Ph.D. Candidate, Department of Mechanical Engineering. Research partially supported by NASA and AFOSR.
[†]Associate Professor, Department of Aeronautics and Astronautics.

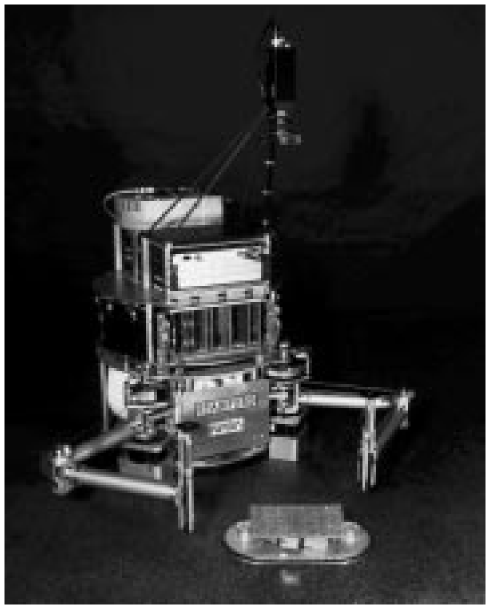
The main benefit is not that it maximizes the connections to neurons ratio, but instead that when combined with a systematic weight pruning procedure, it allows a more flexible use of layering. There has been a recent trend in using not one but two hidden layers; this is a generalization of the trend.

This general architecture makes full use of the backpropagation algorithm, while still allowing the use of modifications, such as the use of FIR connections in place of weights or backpropagation through time [3] [4]. Figure 1 shows 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.

In this application, the extra connections are found to be useful when coupled with a procedure to control over-fitting. In particular, the 3x4 matrix in the upper right corner of the weight matrix provides direct linear information flow from input to output (sigmoids are used only for the outputs of hidden neurons), and the 3x3 upper triangular matrix in the lower right corner provides communication between outputs. While these functions could be provided with processing components in series or parallel with the network, the fully-connected architecture provides a seamless integration of these capabilities.

There are two objectives in this paper. First, the advantages of a fully-connected architecture are demonstrated by applying it to a specific thruster mapping problem characteristic of a space robot. A procedure for handling the over-fitting issue is also demonstrated in this context.

The second objective is to demonstrate experimentally that the advantages are realizable. This is done by performing experiments on a mobile robot which operates in two-dimensions, using air-cushion technology to simulate the drag-free and zero-g characteristics of space. This robot, shown in Figure 2, is a fully self-contained 2-D model 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. The three degrees of freedom (x, y, θ) of the base are controlled using eight thrusters positioned around its perimeter [5].



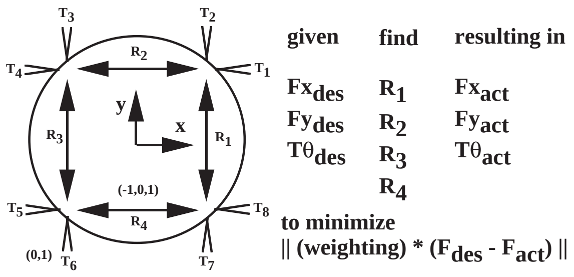


Figure 2: **Stanford Free-Flying Space Robot, Thruster Mapping Problem Definition**

An overview of the thruster mapping problem is presented first to provide a basis for describing the development and demonstration of the benefits of applying a fully-connected neural network.

2 Thruster Mapping Problem Definition

The overall objective is to use a set of eight full-on, full-off air jet thrusters to approximate a continuous-valued force vector that is commanded by the position/attitude control law of the mobile robot. The neural network determines the combination of thrusters to fire that will generate a (normalized) resultant force vector as close as possible to that commanded. Note that the control design "ignored" the restrictions of on-off actuators, and assumed that feedback would compensate for errors due to on-off thrusters.

The right side of Figure 2 shows the eight (0,1) thruster locations, $[T_1, T_2, ..., T_8]$, as well as the equivalent set of four (-1,0,1) reaction forces, $[R_1, R_2, R_3, R_4]$. 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 thruster values, $[T_1, T_2, ..., T_8]$, that minimizes some cost function.

In general, there are 256 possible thruster firing combinations ($2^8 = 256$) that need to be searched to find the optimum. Fortunately, elimination of redundant combinations, as well as the use of forward and inverse transformations to exploit symmetries in the system reduce the search space to 11 [5].

The cost function employed in evaluating the search was a weighted sum of squared errors in the forces achieved. The weighting factors were the effective mass/inertia through which each thrust acted. Consequently, the performance index attempted to match resulting accelerations rather than absolute force values. That is $J = F_{err}^T \cdot Q^2 \cdot F_{err}$ where J is the cost, $F_{err} = F_{des} - F_{act}$, and Q is a diagonal matrix with $[1/M \; 1/M \; r/I_\theta]$ on the diagonal. The thruster firing pattern that minimizes this cost defines the "optimal" mapping which the neural network will emulate.

3 Development of Network

Network training was performed using supervised learning with the backpropagation algorithm to emulate this "optimal" mapping. Batching, and adaptive momentum and learning rate were used to speed up learning. Although these and other modifications to standard backpropagation yielded significant increases in learning performance, they are not the focus of this paper and will not be discussed further.

Careful selection of a pseudo-random training set was found to be helpful in representing the whole problem within a concise training set. A larger, separate test set was formed to evaluate performance of the network, but was not used for training purposes.

3.1 Value of Linear Feed-Through Connections

One of the advantages associated with the fully-connected network is that it provides the ability to implement a linear mapping between the input and output directly. This is advantageous for two reasons. First, if the optimal mapping has a strong linear component, then providing a portion of the network where this can be implemented directly can improve the rate of learning.

Second, if a linear approximation to the solution is known *a priori*, this mapping can be precomputed and loaded into the network before training begins. In this way, the remainder of the network can be used only to implement the nonlinear correction to the *a priori* estimate. This will also increase the rate of learning. Note that the weights associated with the linear component can be frozen or included in the learning depending on the confidence associated with the *a priori* estimate.

For example, in the thruster mapping problem considered here, a reasonable linear approximation to the map can be 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 incorporated this *a priori* knowledge by putting the pseudo-inverse linear solution into that sub-matrix, as an initial condition for the weight matrix.

Figure 3 compares learning histories (thruster mapping error on the training set) for three cases in the thruster mapping example, each with 5 hidden neurons. Benefits of high initial learning rate and enhanced functional capabilities of the the fully-connected network can be seen. The fully-connected network initially learns very quickly, due to the weight gradient being instantly available due to the direct connection of inputs to outputs. For the 3-5-4 layered network, the hidden neurons must first become sufficiently activated to get a strong weight update signal, and it is easy for highly activated neurons to attenuate a backpropagating error signal. The fully-connected network with the *a priori* solution built in provides the best early performance.

After the startup phase, the layered performance surpasses that of the fully-connected, due to the reduced number of parameters, and simplified search space; however, while the layered network performance stops

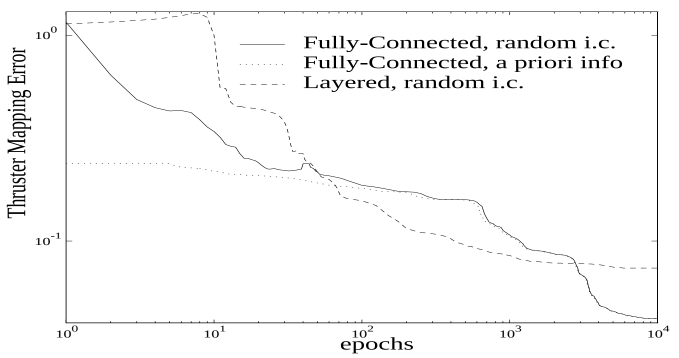


Figure 3: **Training History, Fully-Connected versus Layered**

improving after 1000 epochs, the fully-connected network eventually makes use of its greater functionality to surpass the performance of the layered network.

Observing the weight activation levels during training, the feed-through connections quickly converge, then the others gradually fall into place. By freezing the feed-through weights after an initial learning stage, the "moving target" problem is reduced. This refers to the weights changing directions and back-tracking throughout training while the network approaches a final solution. While this is not necessarily bad, it can slow down learning. The cascade-correlation algorithm takes this concept to its extreme–adapting weights for one neuron at a time, then freezing them before adding the next neuron [6].

3.2 Value of Input and Output Interconnections

The above advantages of the fully-connected network result from direct connections between the input and output layers. Advantages also result from connections between outputs (inputs). The weights associated with these are shown as the upper-triangular 3x3 (2x2) matrix in the lower right (upper left) corner in Figure 1. These weights allow one output to excite or inhibit a higher-numbered output.

The utility of this can be seen in the thruster mapping problem. If the network were to have an output (0,1) for each of the eight thrusters, and during training, a penalty were put on gas use, then the network could use this segment to allow the firing of one thruster to prohibit the firing of the opposite thruster (which would provide zero net thrust and waste fuel). This can be programmed manually prior to learning if observed. Note that the ability to implement this communication between output neurons is unavailable in a layered network.

Another example is the ability to avoid producing a bad result which is the average of two equally good results: if [1 1 -1 -1] and [-1 -1 1 1] both minimize the cost, they may be equally likely to activate when that force is requested, resulting in [0 0 0 0]. The output interconnections would allow the network to use the first output to send it to either of the acceptable solutions, and avoid the ambiguity.

Cross-talk between all outputs would be beneficial, but backpropagation is limited to uni-directional information flow. This may make it important to select carefully the ordering of inputs and outputs. If more complicated, non-linear communication is desired, extra neurons may be placed between individual output or input neurons.

3.3 Architecture Selection to Avoid Over-fitting

The above has shown the potential value of the extra connections associated with a fully-connected neural network, but an important issue that must be addressed is the rapid increase in network parameters with each additional hidden neuron. A single-layered network with i inputs, h hidden neurons, and o outputs has $(i + o) \cdot h$ weights, while a fully-connected network has $((i + h + o) \cdot (i + h + o - 1)/2)$ weights, not counting bias weights. The high number of parameters, while increasing functionality, makes the network susceptible to over-fitting.

It is common that during training, performance on test and training sets will improve until a certain point is reached when the network is no longer finding a better fit to the general problem, but is simply fitting the particular data set. 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 needs a faster solution than this brute-force approach.

The task is to find out which connections are useful and build a network using those weights, and no more. The network architecture selection could be performed manually. For this problem, a network with feed-through connections, weights corresponding to a 3-5-4 layered network, and output interconnections, would probably work well, and not be susceptible to over-fitting, given a good training set.

This heuristic approach may overlook some valuable extra connections, and may still result in over-fitting, so some sort of systematic network pruning technique is required. One used successfully involves the addition of a complexity cost term to the total cost function, as first proposed by Rumelhart [7] [8]. 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. Scale factor, w_0, effectively sets the cutoff point for weights, and λ selects the relative importance of complexity cost versus performance cost. A similar cost function was used to eliminate entire neurons: if the collection of weights into and out of a certain neuron are small, those weights will be reduced.

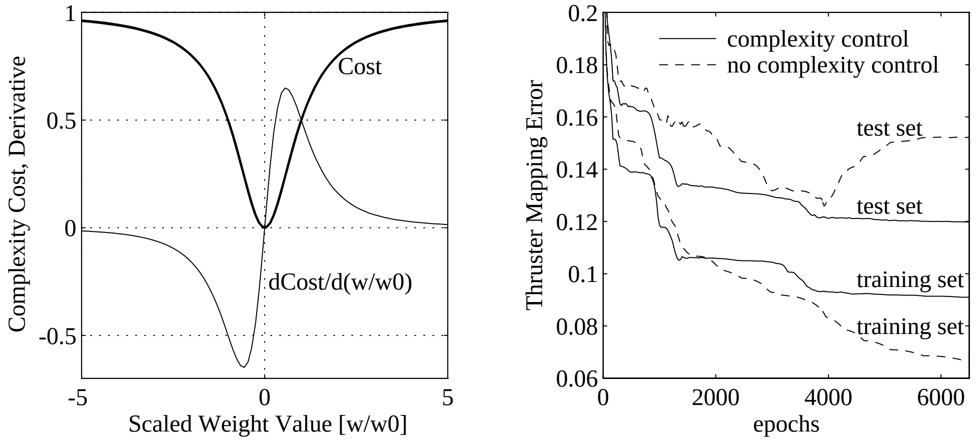


Figure 4: **Complexity Control Function, Effect on Network Performance**

The complexity control function is plotted in Figure 4. Very small weights cost little. Large weights (indicating that they contribute significantly to the network's function) cost λ, but the gradient is small, so there is little incentive to decrease them. Small weights (probably do not significantly help performance) must fight a strong gradient and are likely to be driven to zero. In contrast, a cost function like $\lambda \cdot (w_s^2)$ would be simpler, but it would reduce all the weights, whereas this one only reduces the relatively small weights, leaving the useful ones unrestrained.

Training histories for a fully-connected network with 5 hidden neurons (78 weights) are also shown in Figure 4. The training set has 80 patterns, and the test set 1000. Without any sort of 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, this problem is controlled, as performance histories on test and training sets no longer diverge.

4 Experimental Results

The robot gets position feedback from an overhead "global" vision system. A pair of overhead CCD cameras detect LED's on top of the robot, and an off-board vision processing board computes the robot position and velocity. This information and command signals are transmitted over a wireless Ethernet data/communications link to the vehicle controller on-board the robot.

The neural network thruster mapping component is implemented on the on-board Motorola® 68040 processor, as is the rest of the control system. Running at a 60 Hz sample rate, the calculations use only a small fraction of the available processing power.

Figure 5 shows robot base position during single-axis and multi-axis maneuvers. A simple neural network control component designed using backpropagation through time [2] [4] is used in conjunction with the neural network component described in 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). To avoid clutter, only the robot's actual (x, y) position is plotted.

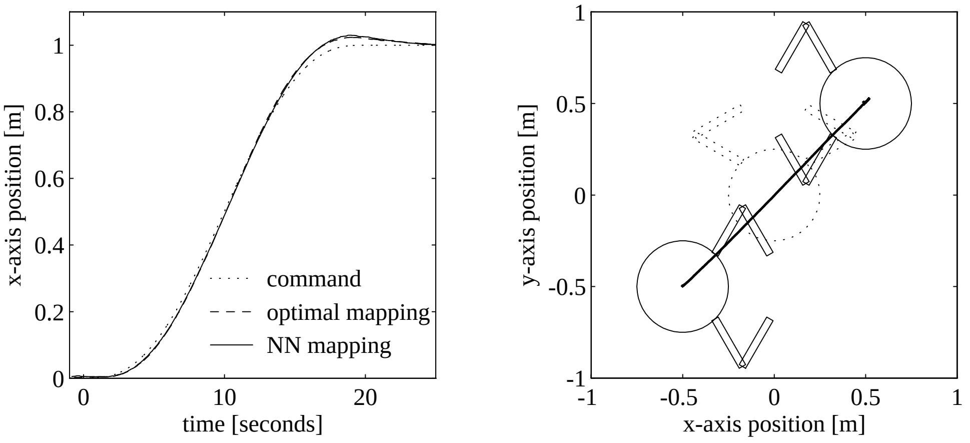


Figure 5: **Experimental Results, Single-Axis Maneuver, Multi-Axis Maneuver**

5 Summary and Conclusions

This paper has described initial steps in development of a neural network control system for a laboratory-based model of a free-flying space robot. A fully-connected neural network, aided by a systematic complexity control scheme, was trained with supervised learning to emulate a complex thruster mapping function typical of spacecraft. The increased connectivity afforded by the layer-less, fully-connected architecture was found to yield specific benefits in this application.

The neural network thruster mapping component was implemented on a 2-D laboratory model of a free-flying space robot. Combined with a simple neural network control component trained using backpropagation through time, it provided accurate tracking during multi-axis trajectories.

References

[1] 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.

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

[3] E. Wan. Temporal backpropagation for FIR neural networks. In *International Joint Conference on Neural Networks*, pages 575–580, San Diego CA, June 1990.

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

[5] Marc Ullman. *Experiments in Autonomous Navigation and Control of a Multi-Manipulator Free-Flying Space Robot.* PhD thesis, Stanford University, Stanford, CA 94305, January 1993.

[6] Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In D.S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*. Morgan Kaufmann Pulishers, 1990.

[7] David E. Rumelhart. Learning and generalization. In *Proceedings of the IEEE Int. Conf. Neural Networks*, San Diego CA, 1988. (plenary address).

[8] 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.