Proceedings of the World Congress on Neural Networks,
San Diego, California, June 1994.

Backpropagation Learning for Systems with Discrete-Valued Functions

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

Abstract

Backpropagation is a powerful learning algorithm, but its restriction to continuously differentiable functions limits its use in many applications. One important example is training a multi-layered neural network of hard-limiting units (signums instead of sigmoids). Another example is a control system that uses discrete-level actuators, such as our free-flying space robot model equipped with on-off gas thrusters.

A new technique is presented that extends backpropagation to allow for discrete-valued functions. Each signum that exists at run-time is temporarily replaced with a sigmoid during training, and noise is injected at the input to the sigmoid. The noise prevents the use of the smooth transition region of the sigmoid as the primary means of solution. The effect is that the sigmoid outputs are close to hard-limited during training so there is not a significant performance reduction when the signums are re-introduced at run-time. The use of differentiable approximating functions allows fast learning due to gradient-based optimization. The noise does not corrupt the gradient estimation algorithm, so no modifications are needed on the backward error propagation.

The viability of this method is verified by applying it to the training of networks with hard-limiting units as well as a complex on-off thruster control problem associated with our free-flying space robot.

1 Introduction

1.1 Optimization with discrete-valued functions

Optimization methods that use gradient information often converge much faster than those that do not. Use of the backpropagation algorithm (BP) [1][2] to get this gradient information for training neural networks (NNs) 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; other systems with discrete-valued inputs and outputs; and neural networks built with signums (aka 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.

1.2 Related research

Learning algorithms for single-layer networks date back to 1960, with Widrow's ADALINE [3] and Rosenblatt's Perceptron [4]. Unfortunately, neither of these methods extend directly to multiple layers.

MADALINE Rule I was a two-layer network (one hidden layer) that had a trainable first layer, but the second layer was a fixed logic operation, such as OR, AND, or MAJ (majority) [5]. In MADALINE Rule II, Winter [6] used a heuristic approach which had limited success at training a two-layer network of hard-limiters (ADALINEs). These methods may be classified as "error-correction rules" rather than "steepest-descent rules" (gradient-based) [3].

In recent research aimed at using gradient-based learning for multi-layer signum networks, Bartlett and Downs [7] use weights that are random variables, and develop a training algorithm based on the fact that

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

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.

Another method 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. This is similar to the original ADALINE, where the neuron was trained on its linear output, but in operation, this output passed through a signum function [3]. This method may work in cases where the behavior of the system with sigmoids is close enough to that of the real system; however, this assumption is very often unreliable.

1.3 Outline of paper

There are three major sections in this paper. In Section 2, the technique of learning with noisy sigmoids is explained and the training algorithm is derived. In Section 3, to demonstrate the usefulness of the method for training multi-layered networks of hard-limiters, it is applied to two different problems. In Section 4, to demonstrate application to a complex control problem, it is applied to a thruster-mapping problem involving eight on-off thrusters controlling a 3-dof free-flying space robot.

2 Backpropagation Learning with Noisy Sigmoids

2.1 Training algorithm

We introduce the method of noise injection by applying it to the training of a single hard-limiting neuron, as shown in Figure 1. This neuron could be trained with the ADALINE or perceptron learning rules, but those methods do not extend to multiple layers, while this one does.

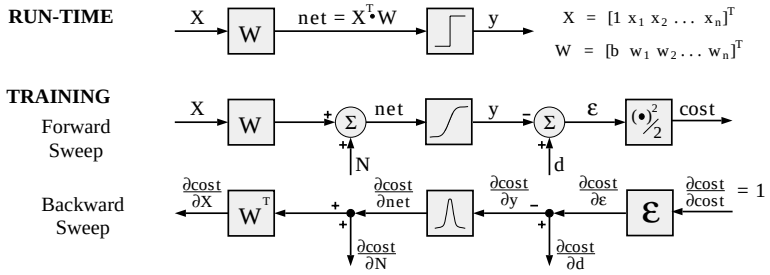


Figure 1: **Training Algorithm**

During training, replace discontinuous signums with sigmoids, 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 1 shows the neuron as it appears at run-time, a dot-product and hard-limiter. For simplicity in bookkeeping, the input, X, and weight, W, vectors are augmented to include the threshold bias for the output function. The next two diagrams show the neuron during training, where the signum 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 weight-update algorithm.

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. While the mechanics of the backward sweep are identical, different weight updates result because the forward sweep resulted in a different error.

Note that the noise injection does not corrupt the calculation of $\partial\text{cost}/\partial W$ (just as the desired signal does not). Using an unmodified backward sweep is not only the simplest thing to do, it does precisely the right calculations for estimating the weight gradient.

What makes this method useful is that as the noise level increases to cover the sigmoid's transition region, adaptation with the resulting $\partial\text{cost}/\partial W$ leads to a set of weights that work well for the signum network. To summarize, the training algorithm is:

- 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

2.2 Why it works

Without addition of noise, the network may train using 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. The goal of noise injection is to move neuron activations away from the transition region during training, so round-off error will be small when the discontinuous functions are replaced.

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. Figure 2 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.

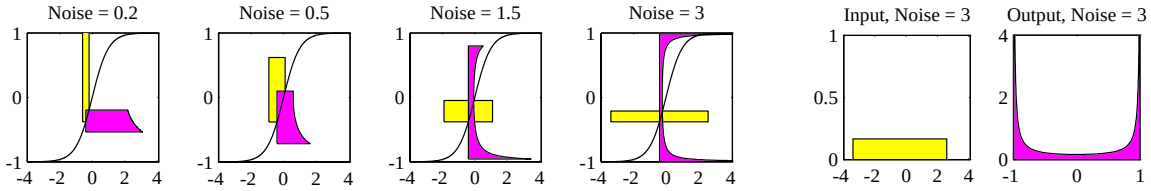


Figure 2: **Effect of Input Noise Level on Sigmoid Output Distribution**

Lightly-shaded region represents the sigmoid input probability distribution (in this case, $-0.3 +$ noise). Darkly-shaded region is the sigmoid output distribution (from -1 to 1), plotted horizontally to correspond to the sigmoid plot. Each distribution has an area of 1. As noise level increases, and the input distribution spreads out, the sigmoid output approaches that of a hard-limiter, while remaining differentiable. At right, input and output distributions are plotted separately.

2.3 Extensions, application considerations

2.3.1 Selection of noise level

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 be gradual in increasing the noise level - slowly push the outputs from the linear region to the hard-limits, rather than all at once. However, since all the experiments presented here had a single layer of discontinuity, no such gradual increase was required.

For training networks with simple bi-level sigmoids, once the noise reached a sufficient level (roughly 0.5 and 3 in two different applications), there was no degradation if it were increased beyond that level. The only possible drawback is the attenuation effect mentioned above. The required noise level varies in different applications depending upon how sharp the decision boundaries would be with no noise (i.e. if it's a sharp sigmoid to begin with, not much noise is needed to force it off the transition region).

When multi-level sigmoids are used, as seen in Figure 9, there *is* an upper limit to noise level. Too much noise may cause the individual sigmoids to overlap, which in this example would blur out the middle level.

2.3.2 Discrete-valued functions other than bi-level signums

If adapting a system that contains discrete-valued functions that are not simple Heaviside step functions, the method may work if a continuously differentiable approximating function is used. For example, a function whose output can take on multiple discrete values may be approximated by combining multiple sigmoid functions. For the thruster mapping problem described in Section 4, the thruster can take on three states: forward, off, or backward. Two bi-level (-1,1) sigmoids were summed to produce a tri-level (-1,0,1) sigmoid.

2.3.3 Batch-learning

The randomness introduced with the addition of noise could make learning slow because of the reduction in signal-to-noise ratio in the weight gradient estimation. 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"). Freezing the training set and noise set defines a fixed deterministic cost hyper-surface. With a fixed cost function, on-line tuning of momentum and learning rate can be applied to dramatically improve convergence rate.

2.3.4 Optimization of discrete-valued parameters

Another area where this method has potential is for optimization problems that have discrete valued parameters. For example, a design optimization problem where the task is to select the right DC motor, pipe diameter, or gear ratio from a finite set of discrete-valued options. It is expected that this method will extend well to this family of problems [8].

3 Application to Training Multi-Layer Signum Networks

In this section, this method is shown to extend to multiple layers of hard-limiting units with no modification. Figure 3 summarizes the method; during training, replace each hard-limiter with a sigmoid and zero-mean independent noise source. Note that the sharpness of the sigmoids does not matter *at all* here, since the sharpness factor simply multiplies the weights, and the weights are adapted.

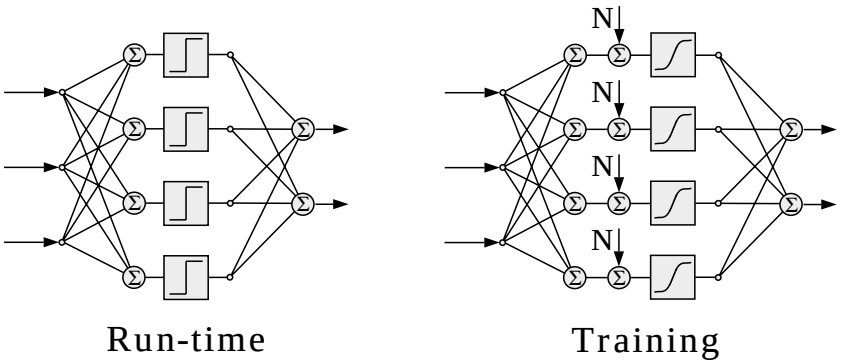


Figure 3: **A multi-layer signum network, seen at run-time and during training**

In the first test, an adaptive $3-5-4$ signum network is trained to emulate the input-output mapping defined by an independent, fixed, $3-10-4$ sigmoidal network. Fewer hidden neurons are used in the adaptive network to ensure that overfitting will not introduce unnecessary complications. The $3-10-4$ network's fixed weights were randomly chosen between -2 and 2.

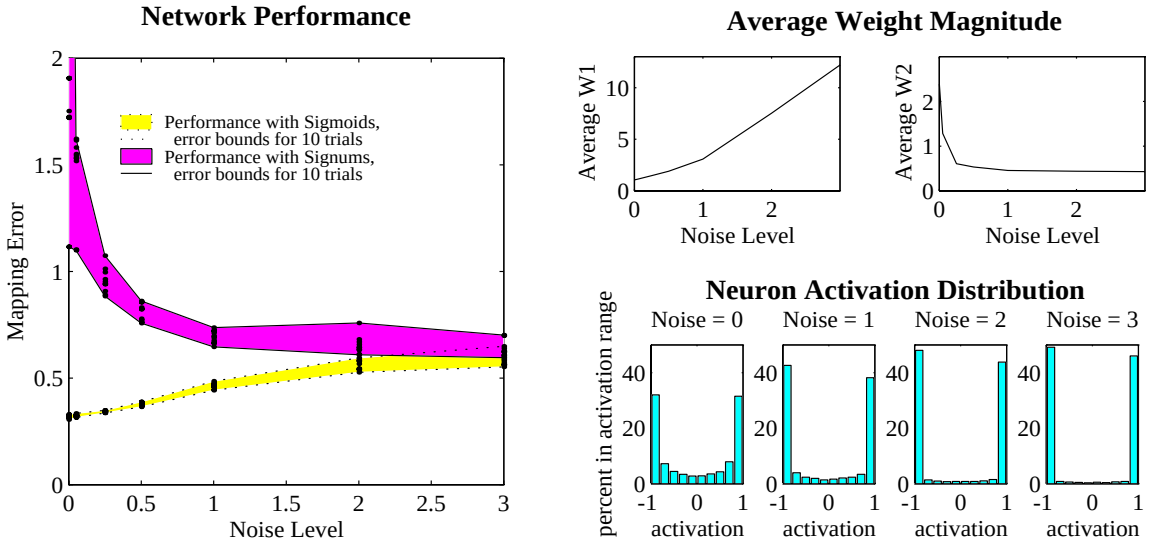


Figure 4: **Direct training of a multi-layer signum network, NN-generated training set**
Left: with higher noise levels, performance on the noisy sigmoidal network approaches that of the signum network, indicating that the noisy sigmoid is a valid (and differentiable!) approximation for the signum. Right: As noise increases, the network adapts to sharpen its sigmoids, causing the first layer weights to increase, and the sigmoid output distributions to approach hard-limiters. Activation distributions were collected over the whole training set, with no noise added.

Performance is shown in Figure 4. Each dot on the graph represents the final performance after a full training run (10,000 epochs or until a local minimum was reached). Seven values for noise level were chosen,

and ten different network initial conditions were used at each noise value. With no noise, performance is good for the sigmoidal network, but when the signums are reintroduced at run-time, the error increases dramatically. One point is off the graph at an error of over 6 units. As noise increases, performance on the sigmoid network decreases, as expected, but the signum-network-performance improves, and approaches the sigmoid-network-performance. The weight magnitude and neuron activation distribution plots confirm that as noise increases, the noisy sigmoids behave like hard-limiters. Note that these activation distributions could not have been achieved by manually increasing the sharpness of the sigmoids - this would have had *zero* net effect since the network would adapt the first layer weights to *exactly* counteract the sharpness increase.

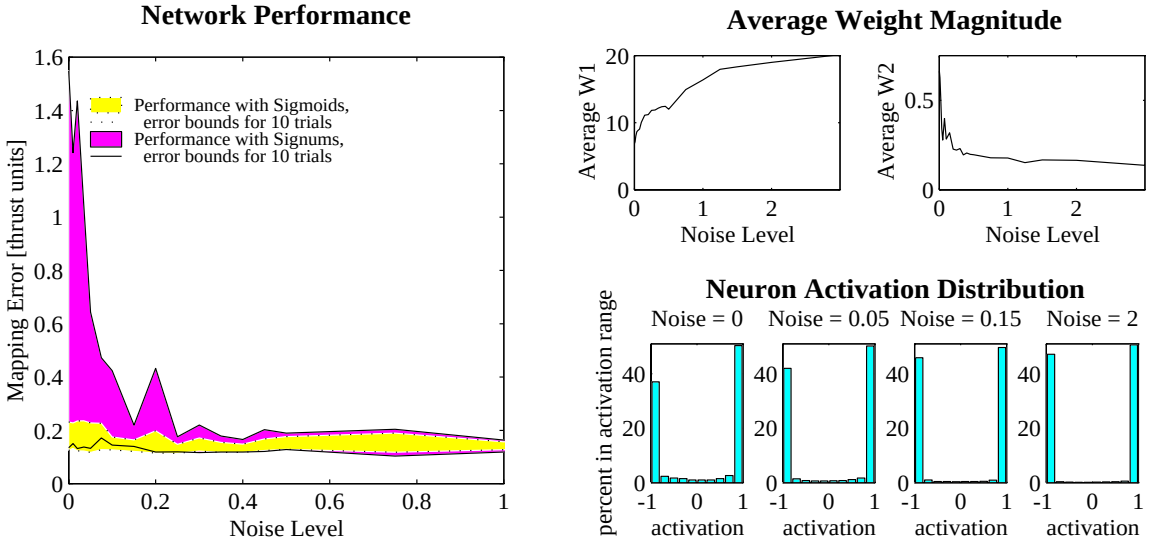


Figure 5: **Direct training of a network of hard-limiters to emulate optimal thruster mapping**

In the second application, the hard-limiting network is trained to emulate the optimal thruster mapping, which will be described in detail in the next section. For now, this mapping is used as an independent second test of the method. A similar dramatic improvement in hard-limiting performance occurs as noise increases past about 0.5. It is not shown on the plot, but good performance is obtained at least up to a noise level of three. The training set for this mapping represents continuous values being mapped to discrete values, so the first-layer weights are high (indicating sharp decision hyper-surfaces), even for $noise = 0$.

4 Application to Robot Thruster Control

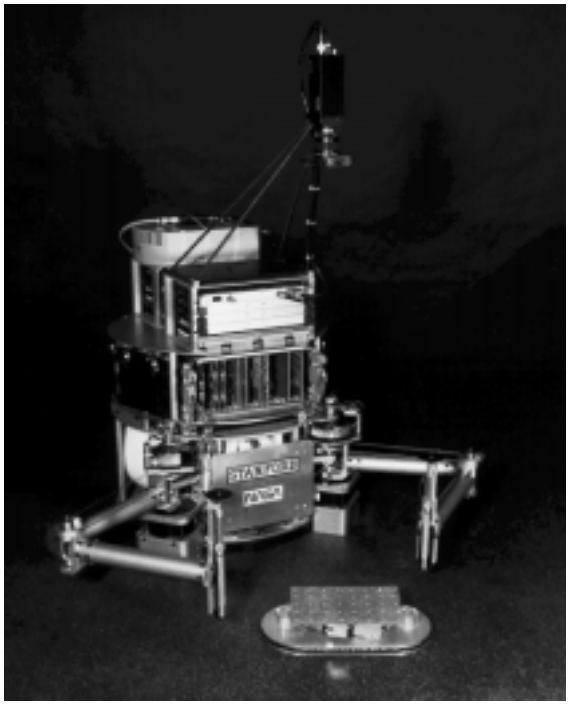
4.1 Robot Description

Experiments are performed using a mobile robot, shown in Figure 6, that operates in a horizontal plane, using air-cushion technology to simulate the drag-free and zero-g characteristics of space[9]. The three degrees of freedom (x, y, θ) of the base are controlled using eight thrusters positioned around its perimeter, shown in the upper right of Figure 6. 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 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.

4.2 Indirect training, Application of noisy sigmoids

Three different techniques used to solve this thruster mapping problem are summarized in the lower right of Figure 6. The first implementation used an exhaustive search at each sample period to find the thruster pattern minimizing the force error vector [9]. 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 directly to emulate the optimal mapping produced by the exhaustive search, described in Section 3 and in [10] [11].



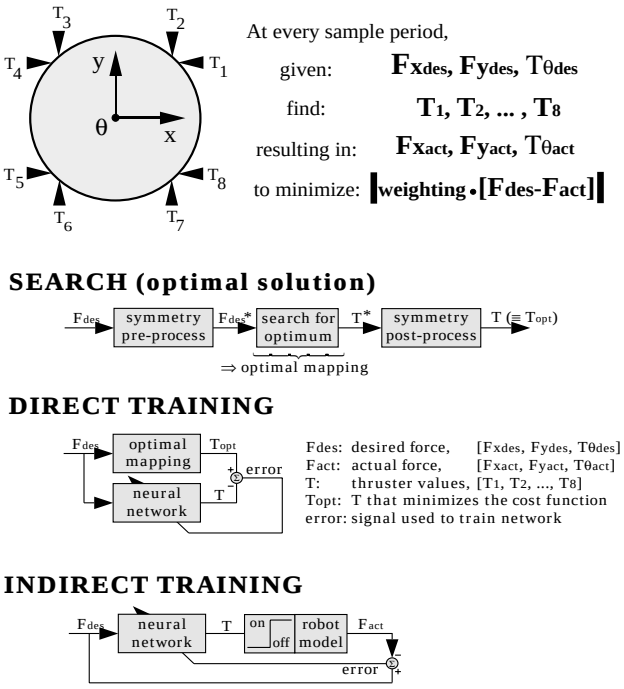


Figure 6: **Free-flying robot, Thruster mapping problem definition, Solution methods**

Left: The robot 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. Right: [Desired forces ⇒ Thruster values] mapping: problem definition, solution methods. The on-off thrusters and coupling between forces and torque make this problem difficult.

The third method, indirect training, is presented here and shown in Figure 7. In this case, the optimal mapping is not used, and the NN must learn the mapping through experimentation with the plant model. This requires back-propagation of error through the discontinuous thrusters, which motivated development of the noise injection method presented in this paper.

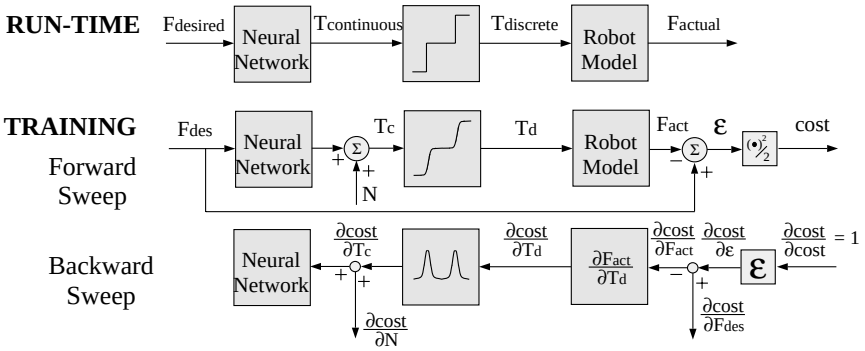


Figure 7: **Thruster mapping, indirect training method**

Figure 8 shows the result of training with two differentiable thruster models. During training with the continuous thruster models, the NN produces a mapping with a very low error, but when the signums are replaced at run-time, the error is large. This is 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.

Figure 9 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 8, but as noise increases, performance approaches that of the

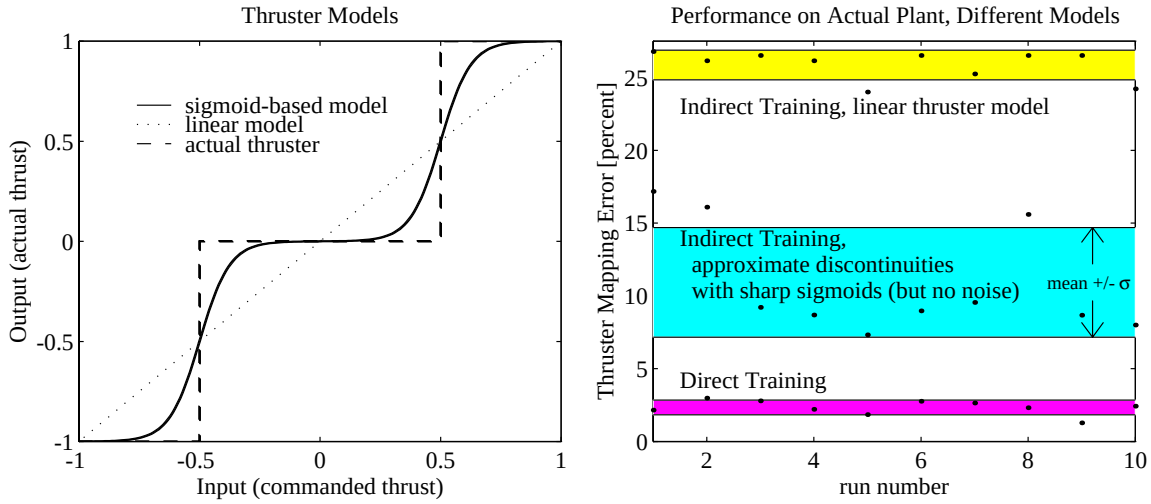


Figure 8: **Results of indirect training, two differentiable thruster models**

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.*

network trained directly (emulating the optimal mapping). The direct-performance represents a lower bound set by the functional complexity of the $3 - 10 - 4$ layered network. The best noise value in this application seems to be around 0.15, and the resulting noisy sigmoid is shown in the left side of Figure 9. 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 the "off" region of the sigmoid becomes blurred) as expected, but the method is fairly robust to the noise value selected.

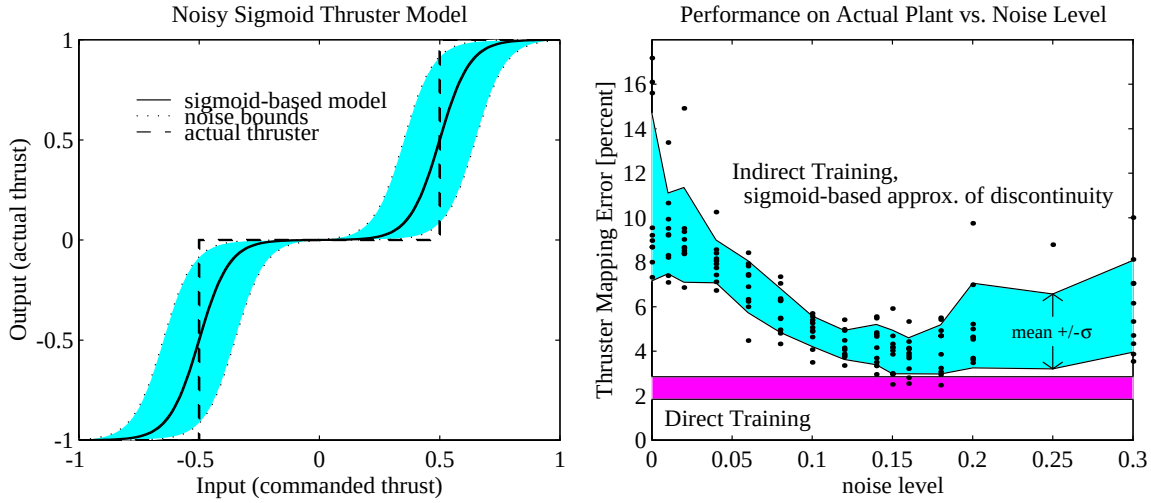


Figure 9: **Results of indirect training, noisy tri-level sigmoid thruster model**

Left: the sharpness (4) *and noise level* (0.15) *for the noisy tri-level sigmoid appear to be intuitively correct. Right: as noise increases, performance approaches that of the network trained directly (emulating the optimal mapping).* $3 - 10 - 4$ *layered networks were used.*

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 region to consistently work well. The results approximate the optimal solution very well, and work when the tri-level sigmoids are replaced with tri-level signums.

5 Conclusions

This paper has described a new technique that allows backpropagation learning to work with systems containing discrete-valued functions, despite the discontinuity that exists between discrete values. The modification to backpropagation is very small, simply requiring sigmoidal approximation of the discrete-valued functions, and careful injection of noise into the smooth approximating function on the forward sweep. The noise injection is critical to ensuring that the noisy sigmoid behaves like a signum during training.

Multi-layered networks of hard-limiters require simpler processing hardware than do multi-layered sigmoid networks. Sigmoid networks are commonly used due to their increased functionality as well as the lack of a reliable training algorithm for signum networks. Multi-layered signum networks have now been successfully trained using this noise injection method in two different applications, clearly demonstrating its usefulness in this area.

Application to a complex thruster control problem, with implementation on a laboratory model of a free-flying space robot, has demonstrated the method's realizability and usefulness for on-off control problems.

In each application, the training behavior in the presence of noise has been well-understood, and the algorithm appears to be relatively robust to the amplitude of the injected noise.

Acknowledgements

The author wishes to thank AFOSR and NASA for their support of this research, and Tim McLain, Dr. Larry Pfeffer, Glen Sapilewski, and Prof. Bernard Widrow for their help reviewing the paper.

References

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

[2] 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.

[3] B. Widrow and M.A. Lehr. 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9):1415–42, September 1990.

[4] Frank Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Spartan Books, Washington, D.C., 1962.

[5] M.E. Hoff, Jr. *Learning Phenomena in Networks of Adaptive Switching Circuits.* PhD thesis, Stanford University, Stanford, CA 94305, July 1962. Tech. Rep. 1556-1, Stanford Electron. Labs.

[6] R. Winter and B. Widrow. Madaline rule II: a training algorithm for neural networks. In *IEEE International Conference on Neural Networks*, volume 1, pages 401–408, San Diego CA, July 1988.

[7] 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.

[8] Timothy W. McLain. Personal communication, 1993.

[9] 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.

[10] 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.

[11] Edward Wilson. Experiments in neural network control of a free-flying space robot. In *Proceedings of the Fifth Workshop on Neural Networks: Academic/Industrial/NASA/Defense*, pages 204–209, San Francisco CA, November 1993.