TOWARDS HUMAN CONTROL STRATEGY LEARNING: NEURAL NETWORK APPROACH WITH VARIABLE ACTIVATION FUNCTIONS

Michael C. Nechyba Yangsheng Xu

CMU-RI-TR-95-09

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

March, 1995

This material is based upon work partially supported under a National Science Foundation Graduate Research Fellowship, as well as a Department of Energy Doctoral Research Fellowship.

Table of Contents

List of Figures

List of Figures

List of Figures

List of Tables

Abstract

Human beings epitomize the concept of “intelligent control.” Despite its apparent computational advantage over humans, no machine or computer has come close to achieving the level of sensor-based control of which humans are capable. Thus, there is a clear need to develop computational methods which can abstract human skill based on sensory feedback. Neural networks offer one such method with their ability to map complex nonlinear functions. This paper is divided into two parts. First, we examine the problem of approximating continuous functions, as is required for dynamic system identification and control. We discuss how the requirements of continuous function approximation differ substantially for those of discrete function approximation. To meet these requirements, we propose to use the cascade two learning architecture, which dynamically adjusts the size of the neural network as part of the learning process. As such, we propose different hidden units to have variable activation functions, leading to faster learning, as well as better function approximation. Second, we apply these methods towards the problem of control system identification, and more specifically, to the problem of identifying and modeling human control strategy. We demonstrate the feasibility of the proposed method in human control strategy or skill learning, and address issues for potentially exciting research in the future. This approach can play a significant role in the development of intelligent machines based on human skill learning, as well as in the intelligent and harmonious coordination of humans and robots.

1 Introduction

Rapid advances in computer technology over the past decade have not been paralleled by corresponding rapid advances in robot capabilities or the development of “intelligent machines.” This disparity is principally caused by the difficulty of formalizing intelligent human behavior and decision making processes into an *algorithmic* framework. Humans manage everyday tasks, such as visual processing, manipulation, and mobility, with relative ease; yet, it has proven difficult, if not impossible, for robots to duplicate that performance adequately. That is, although humans are quite successful at executing these tasks, they are far less successful in describing them in a robust and machine-understandable manner. In effect, human skill remains locked away in the human mind where it is of little use in the development of robot functionality. Thus, effective modeling of human control strategy in a *non-algorithmic* fashion is an essential step towards (1) the development of intelligent robots and the transfer of human skill to those robots, and (2) better coordination and cooperation between humans and robots.

“Intelligent robots” must deal robustly with uncertainty, hostile environments, or complex task scenarios based on multiple sensory inputs. Such robots typically require more complex behavior than simply tracking a pre-programmed trajectory, for example. Furthermore, the required behavior may be sufficiently difficult in nature, so as to invalidate algorithmic or rule-based solutions, which may fail to anticipate every eventuality. For example, although fuzzy control systems [13][14] are appealing due to their linguistic structure, this particular rule-based approach becomes less feasible when the number of inputs (i.e. sensors) is large. In such cases, the robot should “learn” the correct behavior by “observing” the appropriate control strategy from human training examples; that is, the robot should model the human control strategy. This approach, for example, has proven to be quite successful in the autonomous driving domain [19].

In addition to facilitating the development of intelligent robots, modeling of human control strategy also has important applications in improving coordination and cooperation between humans and robots. Traditionally, human-robot interactions have been confined to the “master-slave” paradigm, where the human acts as the “master” teleoperator, while the robot acts as the, often ignorant, “slave.” This is mainly due to the fact that humans have a very good model of the robot in mind, while robots, on the other hand, do not have a very good model of humans. Such a model is necessary, however, if we are to improve the shared, intelligent control of robots, where the human acts as a global supervisor, while the robot acts

as a local protector. In this way, humans and robots can complement and enhance each others performance in a reciprocal manner.

Valuable as modeling human control strategy may be, there are many difficult aspects to this problem. In general, tasks best learned from human training examples are highly nonlinear and time-varying in nature. Furthermore, these tasks may require that part or all of the learning occur in real-time. Some work has been done towards transferring skill from humans to robots in limited task domains. In [19], a car is taught to drive autonomously, using a multilayer feedforward neural network to map camera images of the road ahead to a steering direction. In [3], a neural network learns the deburring task for a machining robot with training data from a human expert. Hidden Markov Models (HMMs) have also been suggested as a possible means for modeling human performance [20]. HMMs show potential especially in higher level abstraction of human skill and performance.

Artificial neural networks, however, have shown the most promise in identifying complex nonlinear mappings, and have found many applications in nonlinear control [1][2][3][9][11][15][16][17][19]. Thus, neural networks are well suited for generating the complex internal mappings from sensory inputs to control actions which humans possess. We are therefore interested in developing a feasible neural network-based method for identifying and modeling human control strategy. To this end, we examine an efficient and flexible neural network architecture capable of modeling nonlinear dynamic systems.

We propose to use the cascade two learning architecture [6], which adjusts the structure of the network as part of the training process. Unlike a fixed network architecture, where hidden unit activation functions are specified before training begins, the incremental addition of new hidden units allows for the hidden units to have variable activation functions. Although sigmoidal activation functions may be best suited for networks with discrete outputs, this is not necessarily the case for continuous-valued outputs. Allowing new hidden units with variable activation functions can lead to better approximations, as well as faster learning with less computation and fewer hidden units.

In this paper, we first provide background information on this new architecture for neural network learning and a theoretical basis for its use. Second, we present detailed simulations characterizing learning speed and approximation capacity for networks with differing activation functions. These results are then applied to the identification of some dynamic systems, including a nonlinear control law for an inverted pendulum system. Finally, we

demonstrate the successful implementation of the learning architecture in identifying human control strategy.

2 Neural Network Learning

2.1 Discrete *vs.* Continuous Function Approximation

Traditionally, the sigmoidal function,

$$sig(x) = \frac{1}{1 + e^{-x}}, \qquad \text{(Eq. 1)}$$

or the symmetric sigmoidal function,

$$sig_{sym}(x) = \frac{1}{2}\left(\frac{1 - e^{-x}}{1 + e^{-x}}\right) \qquad \text{(Eq. 2)}$$

has been the activation function of choice for hidden units in neural network applications. For discrete (i.e. binary) mappings or classification tasks, this choice allows a network to form the sharp divisions required to distinguish between 0 and 1, since it is essentially a smoothed version of the threshold function,

$$f_{threshold}(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases} \qquad \text{(Eq. 3)}$$

Continuous mappings differ, however, in that sharp dividing decision surfaces do not have to be formed, in general. Thus, with respect to functional approximation, there is no *a priori* motivation to choose the sigmoidal function over other nonlinear functions for the activation function of the hidden units. In other words, sigmoidal functions may not form the best basis for continuous function approximation in neural network applications.

Discrete and continuous function approximation differ in one other important aspect. Learning for discrete mappings may be stopped when network outputs fall within some relatively large *range* of the target outputs. For continuous mappings, however, learning must proceed until network outputs and target outputs fall within some much smaller ε. As such, a standard multilayer feedforward network will take significantly longer in converging for continuous function approximations than discrete function approximations of the same order of complexity.

For these reasons, we have chosen a neural network learning architecture which offers two distinct advantages over multilayer backpropagation networks: (1) faster convergence, and (2) an opportunity to employ *variable* activation functions.

2.2 Cascade Architecture

The cascade two learning architecture, developed by Fahlman [7], is similar to the previously developed cascade correlation algorithm. Both cascade correlation and cascade two combine the following two notions: (1) the *cascade architecture*, in which hidden units are automatically added one at a time to an initially minimal network, and (2) the accompanying learning algorithm, which creates and installs the new hidden units [6][7].

In cascade correlation, the learning algorithm attempts to maximize the magnitude of the correlation between the new hidden unit's output and the residual error signal. This covariance measure tends to overshoot small errors, however, and thus is not suitable for continuous valued outputs. The cascade two algorithm corrects this problem by attempting to minimize the sum-squared difference between the scaled unit outputs and the residual error [6].

Training proceeds as summarized below. Initially, there are no hidden units in the network, only the input/output connections. These weights are trained first, thereby capturing any linear relationship between the inputs and outputs. With no further appreciable decrease in the error measure (in cascade two, the sum-squared error), the first hidden unit will be added to the network from a pool of *candidate* units. Using the quickprop algorithm [5], these candidate units are trained independently and in parallel with different random initial weights.

After no more appreciable error reduction occurs, the best candidate unit is selected and installed in the network. Once installed, the hidden unit input weights are frozen, while the weights to the output units are retrained. By freezing the input weights for all previous hidden units, each training cycle is equivalent to training a three-layer feedforward neural network with a single hidden unit. This allows for much faster convergence of the weights during training than in a standard backpropagation network where many hidden unit weights are trained simultaneously.

The process is repeated until the algorithm succeeds in reducing the sum-squared error sufficiently for the training set or the number of hidden units reaches a specified maximum number. Note that each new hidden unit receives as input connections from all previous units,

including all input units as well as previous hidden units. Figure 1 below illustrates how a 2-input, 1-output network grows as 2 hidden units are added.

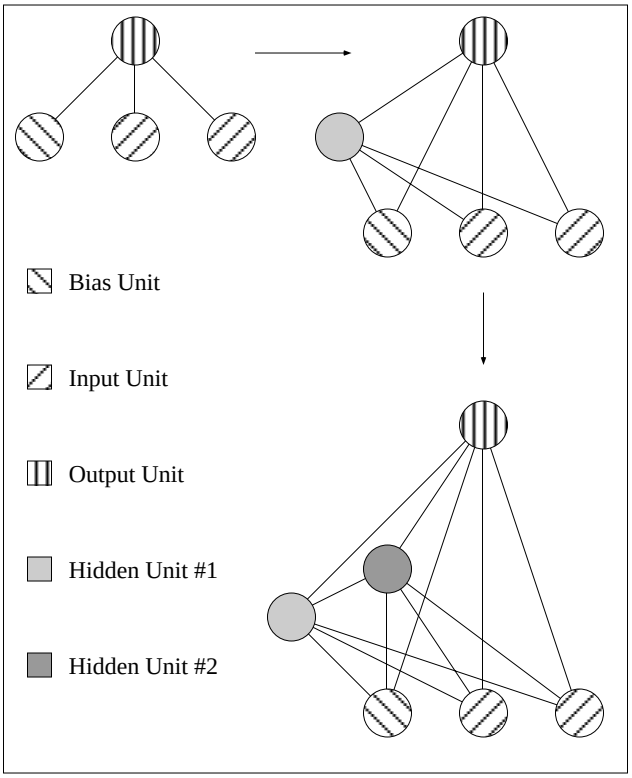

Figure 1: The cascade two learning architecture adds hidden units one at a time as shown in the above diagram. All connections are feedforward.

The cascade two architecture offers several advantages, particularly relevant for the mapping of non-linear continuous-valued functions. First, the algorithm adjusts the architecture of the network automatically, thus obviating the need for *a priori* guessing of the necessary network architecture. Second, the cascade architecture can potentially model higher degrees of nonlinearity with fewer hidden units than might be required in a single or two hidden-layer network. Finally, and perhaps most importantly, the incremental addition of hidden units allows for new hidden units to have *variable* activation functions.

2.3 Variable Activation Functions

In general, function approximation is composed of two parts: (1) the selection of an appropriate functional form with free parameters, and (2) the adjustment of those free parameters as a function of some optimization criterion. For most neural networks used today, the learning process consists of (2) only, since a specific functional form is selected

before learning begins; that is, the network architecture is usually fixed before learning. In this respect, neural networks are similar to other mathematical approximation techniques such as the Fourier series and Least Squares approaches. Neural networks, differ, however, in that the free parameters, i.e the weights, vary nonlinearly with the inputs.

We believe that both (1) and (2) have a place in the learning process. The cascade two architecture by itself relaxes *a priori* assumptions about the functional form somewhat by dynamically adjusting the network size. We propose, however, to further relax these assumptions for the approximating network by allowing new hidden units to have *variable* activation functions.

In the pool of candidate units, we can assign a different nonlinear activation function to each unit. Thus, if the function to be approximated has a strong sinusoidal dependence, for example, it is more efficient to have one sinusoidal hidden unit rather than several sigmoidal units acting together to first approximate that sinusoidal dependence. During candidate training, the algorithm selects for installment whichever candidate unit reduces the sum-squared error of the training data the most. Hence, the unit with the most appropriate activation function at that point during training is selected. For all the simulation results in this paper, we use the eight nonlinear functions described in 1.

Table 1: Nonlinear Functions used as Activation Functions

	Type 1	*Type 2*	*Type 3*	*Type 4*
$f(x)$	$\frac{1}{1+e^{-x}} - \frac{1}{2}$	$\exp\left(-\frac{1}{2}x^2\right)$	$\sin(x)$	$\sin(2x)$
Description	symmetric sigmoid	Gaussian	sine	double frequency sine[a]
	Type 5	*Type 6*	*Type 7*	*Type 8*
$f(x)$	$\cos(x)$	$\cos(2x)$	$J_0(x)$	$J_1(x)$
Description	cosine	double frequency cosine[b]	Bessel function	Bessel function

a. The double frequency sine function allows weights to remain smaller than for the simple sine function.
b. The double frequency cosine function allows weights to remain smaller than for the simple cosine function.

Note that some of the activation functions we use are not monotonically increasing. Although such activation functions might lead to oscillating or even nonconvergent behavior in standard multilayer feedforward networks, this problem does not manifest itself in the

cascade two learning architecture, since each step of the learning process in cascade two requires, at most, backpropagation through one hidden unit at a time.

Figure 2 illustrates the relative frequency of each function type selected by the learning algorithm over hundreds of simulation runs. Some 80 percent of all “variable” function types selected are some form of sinusoidal function (types 3, 4, 5, and 6 in 1). Sigmoidal functions (type 1), however, are selected virtually none of the time. This suggests that at least for continuous function approximation, sinusoidal rather than sigmoidal units may be more appropriate. Therefore, in addition to simulating networks with variable activation functions and sigmoidal activation functions, we also simulate networks with exclusively sinusoidal units.

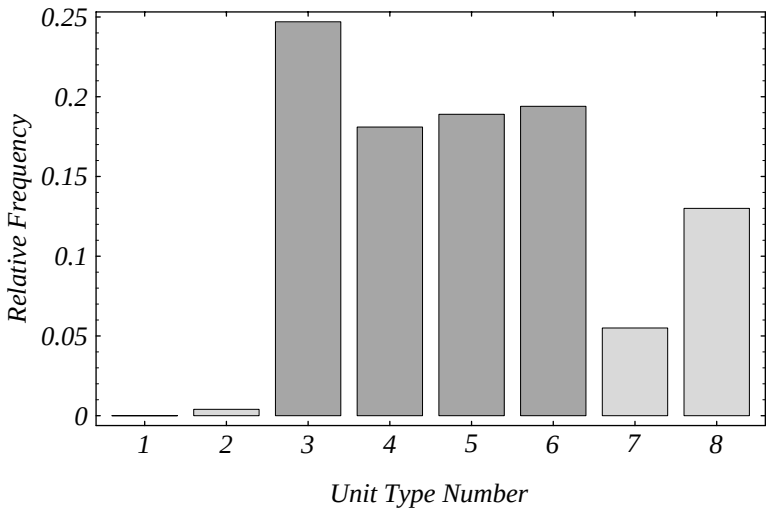

Figure 2: Most (80%) of the “variable” activation functions selected in the learning process were of a sinusoidal type (types 3, 4, 5, and 6).

2.4 Theoretical Properties

Here, we show that this neural network architecture is capable of arbitrary, nonlinear function approximation. Over the past several years, some key results have been established regarding the approximation properties of multi-layer feedforward networks with full connections between consecutive layers.

Kurkova shows in [12] that a feedforward neural network with two hidden layers is sufficient for arbitrary function approximation. In fact, Cybenko and Funahashi have shown separately that a continuous feedforward neural network with a *single* hidden layer and activation functions with continuous sigmoidal nonlinearities can approximate nonlinear mappings arbitrarily well [4][8].

These function approximation theorems can be extended for the cascade two architecture. For now we will assume only sigmoidal units. First we realize that any multilayer feedforward neural network with *k* hidden units arranged in *m* layers, fully connected between consecutive layers, is a special case of a cascade network with *k* hidden units with some weight connections equal to zero. Consequently, existence theorems established for layered feedforward networks follow trivially for the cascade two architecture.

Furthermore, Cybenko shows that sigmoidal functions are not the only possible activation functions which allow for arbitrary function approximation. In fact, there are other nonlinear functions, such as *sine* and *cosine* for example, which are complete in the space of *n*-dimensional continuous functions. Hence, there is no strict theoretical argument for confining the activation functions exclusively to sigmoidal functions.

3 Continuous Function Approximation

In this section, we examine the problem of continuous function approximation with simple one-variable functions. The simulation results presented here serve a two-fold purpose: (1) within the cascade two framework, the simulations demonstrate the feasibility and advantage of variable activation function types over *a priori* specification of activation functions as sigmoids; and (2) the results show that the cascade two architecture outperforms standard multilayer feedforward networks with an equal number of adjustable parameters in both learning speed as well as approximation capability.

For these simulations, we allow a maximum of 150 epochs for training the candidate input weights as well as the installed hidden unit output weights. In each case, a pool of eight candidate units was used. As training data, we generate 1,500 data points, which are uniformly randomly distributed over the domain of interest. Training is halted after 10 hidden units have been installed in each network. For each example, 25 separate trials were conducted. The error plots indicate the median performance of each network among those 25 trials. We refer to a network where all hidden units have the sigmoidal activation function as a “sigmoidal network.” A network where all hidden units have sinusoidal activation functions is referred to as a “sinusoidal network,” while a network with variable activation functions is referred to as a “variable network.” In all cases, output units are linear.

3.1 Example 1: Polynomial Function

For this example, we wish to approximate the following polynomial function,

$$f(x) = x^3 + 0.3x^2 - 0.4x, \; x \in [-1, 1] \qquad \text{(Eq. 4)}$$

Figure 3 plots $f(x)$ over the domain of interest. Figure 4 compares the approximation error for the variable and sigmoidal networks. Figure 5 compares the approximation error for the sinusoidal and sigmoidal networks. Figure 6 illustrates how the average root-mean-squared (RMS) error over all trials decreases as a function of the number of hidden units for each network type. Finally, Figure 7 illustrates the number of epochs of training required as a function of the number of hidden units for each network type.

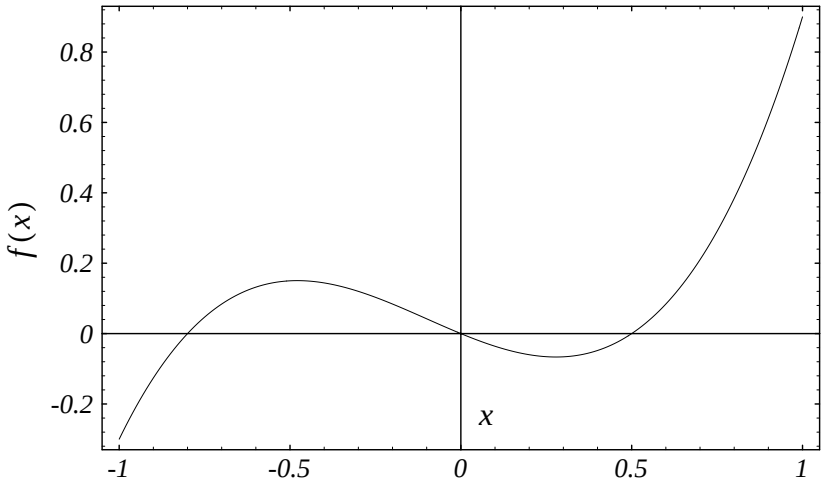

Figure 3: Function to be approximated for example 1.

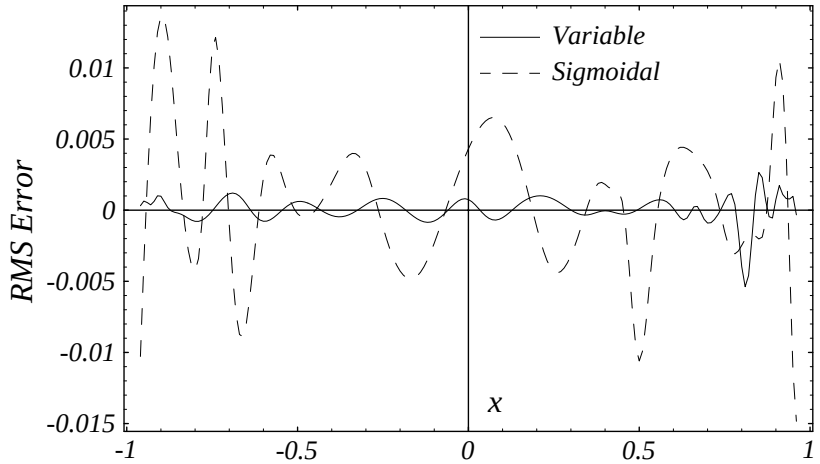

Figure 4: RMS approximation error for the median variable network and the median sigmoidal network.

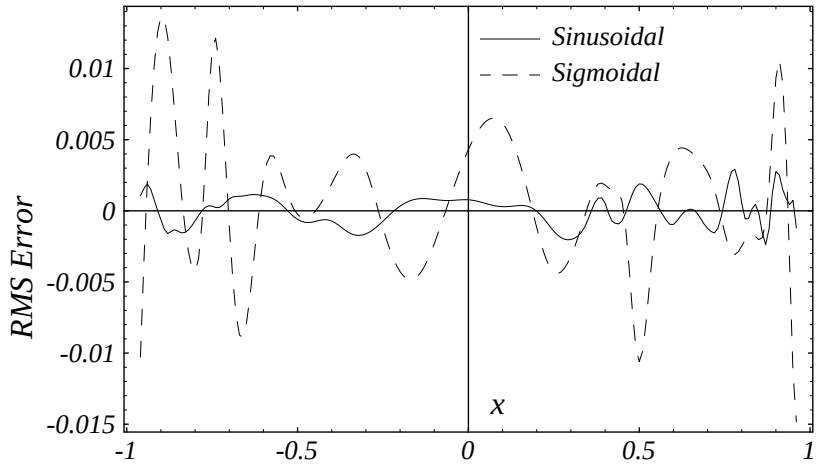

Figure 5: RMS approximation error for the median sinusoidal network and the median sigmoidal network.

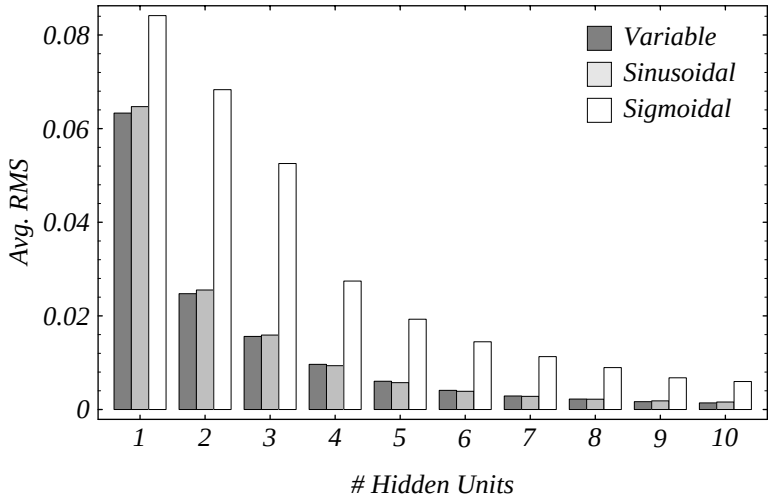

Figure 6: The RMS error decreases much faster in the variable and sinusoidal networks than in the sigmoidal network.

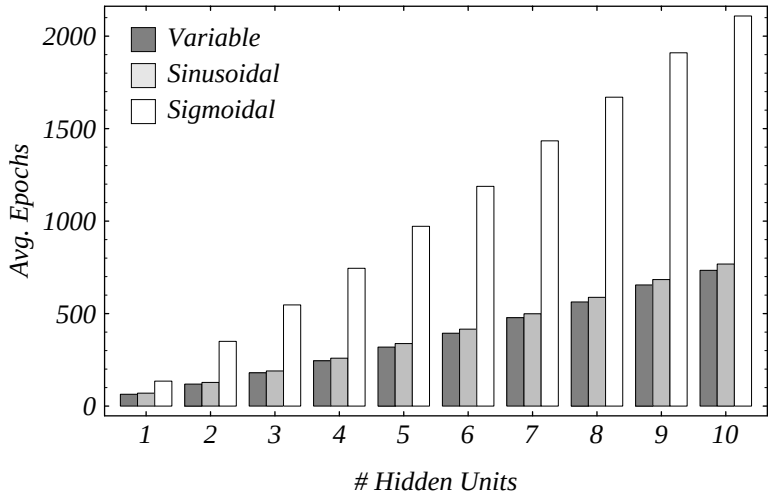

Figure 7: Learning is much faster in the variable and sinusoidal networks than in the sigmoidal network.

3.2 Example 2: Rational Function

For this example, we wish to approximate the following rational function,

$$f(x) = \frac{1}{(1 + x^2)}, \; x \in [-4, 4] \qquad \text{(Eq. 5)}$$

Figure 8 plots $f(x)$ over the domain of interest. Figure 9 compares the approximation error for the variable and sigmoidal networks. Figure 10 compares the approximation error for the sinusoidal and sigmoidal networks. Figure 11 illustrates how the average root-mean-squared (RMS) error over all trials decreases as a function of the number of hidden units for each network type. Finally, Figure 12 illustrates the number of epochs of training required as a function of the number of hidden units for each network type.

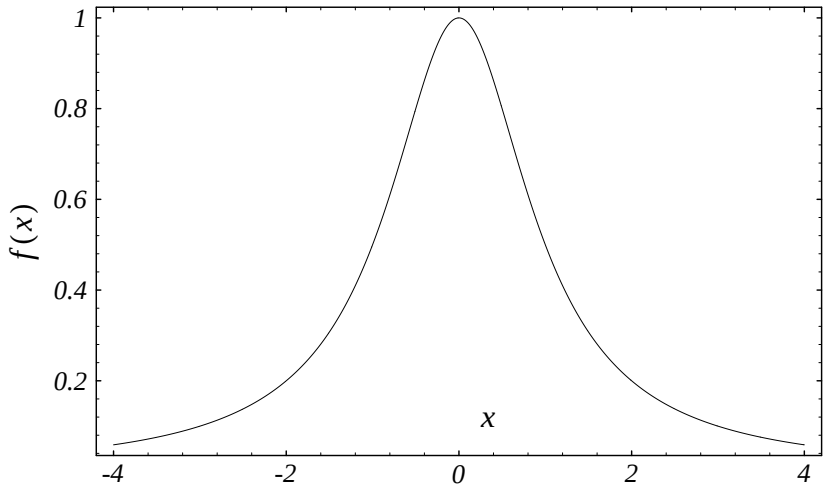

Figure 8: Function to be approximated for example 2.

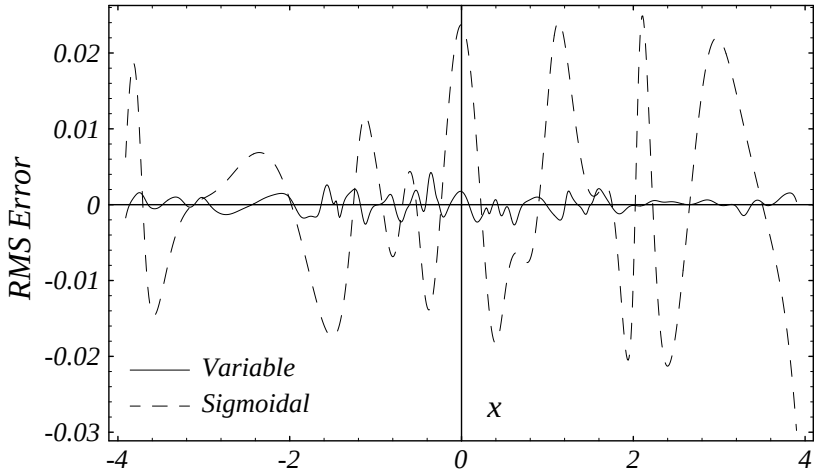

Figure 9: RMS approximation error for the median variable network and the median sigmoidal network.

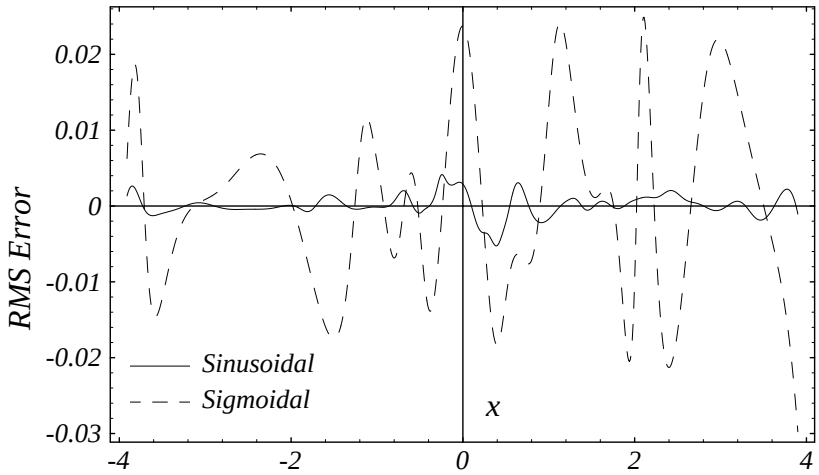

Figure 10: RMS approximation error for the median sinusoidal network and the median sigmoidal network.

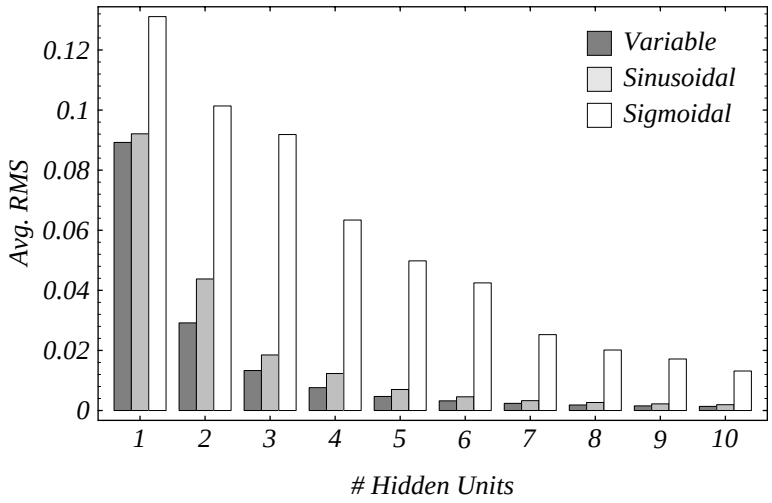

Figure 11: The RMS error decreases much faster in the variable and sinusoidal networks than in the sigmoidal network.

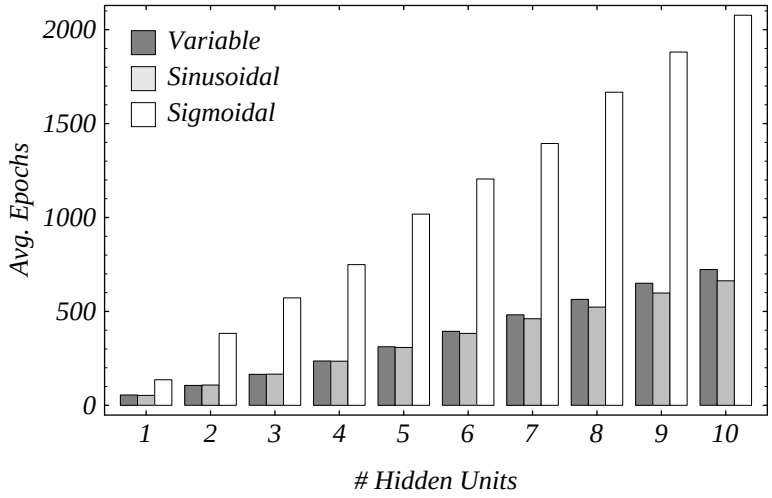

Figure 12: Learning is again much faster in the variable and sinusoidal networks than the sigmoidal network.

3.3 Example 3: Trigonometric Function

For this example, we wish to approximate the following trigonometric function,

$$f(x) = 0.6\sin(\pi x) + 0.3\sin(3\pi x) + 0.1\sin(5\pi x) \quad , \; x \in [-1, 1] \qquad \text{(Eq. 6)}$$

Figure 13 plots $f(x)$ over the domain of interest. Figure 14 compares the approximation error for the variable and sigmoidal networks. Figure 15 compares the approximation error for the sinusoidal and sigmoidal networks. Figure 16 illustrates how the average root-mean-squared (RMS) error over all trials decreases as a function of the number of hidden units for

each network type. Finally, Figure 17 illustrates the number of epochs of training required as a function of the number of hidden units for each network type.

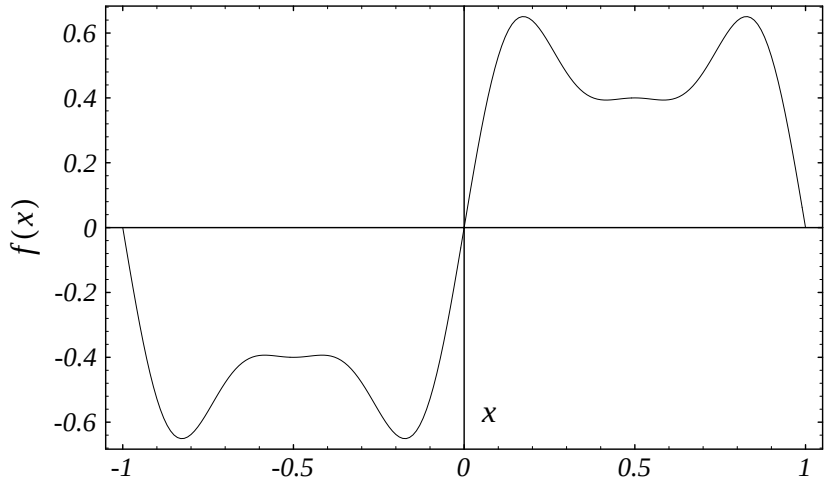

Figure 13: Function to be approximated for example 3.

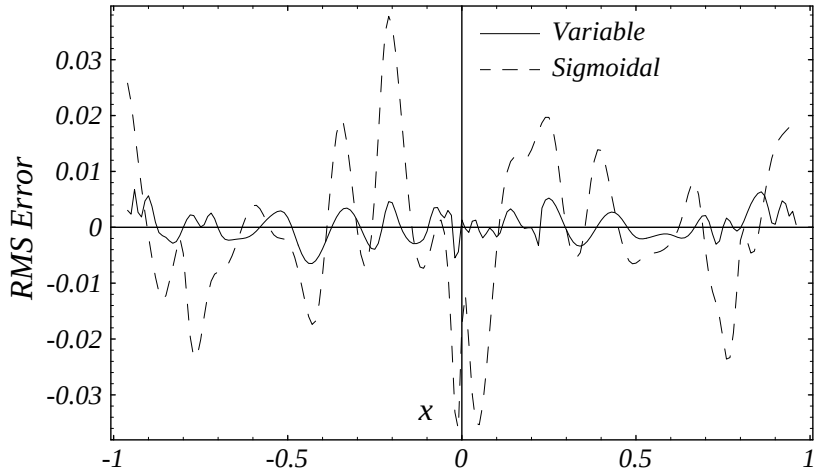

Figure 14: RMS approximation error for the median variable network and the median sigmoidal network.

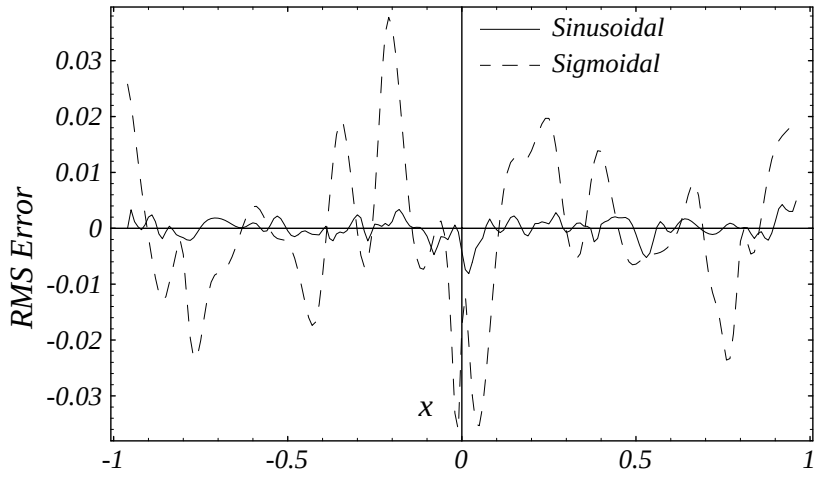

Figure 15: RMS approximation error for the median sinusoidal network and the median sigmoidal network.

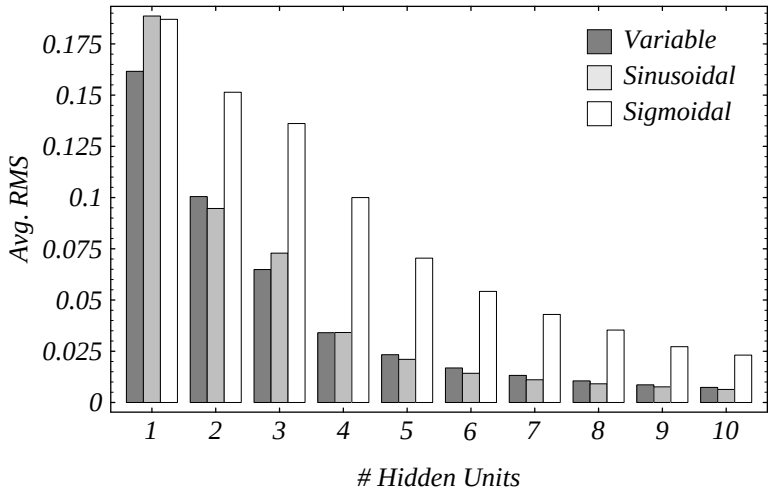

Figure 16: The RMS error decreases much faster in the variable and sinusoidal networks than in the sigmoidal network.

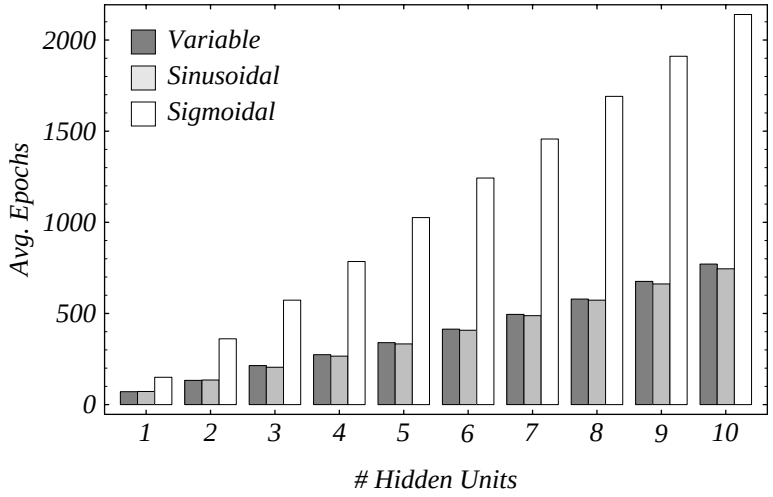

Figure 17: Learning is again much faster in the variable and sinusoidal networks than the sigmoidal network.

3.4 Discussion

2 summarizes RMS statistics for the three examples; 3 summarizes learning rate statistics for the three examples; and 4 compares performance between the sigmoidal, sinusoidal, and variable networks.

Table 2: RMS Error Statistics for Simulation Examples

	Network Type	*Min Error*	*Max Error*	*Mean Error*	*Std. Deviation*
Example 1	Variable	0.0008	0.0023	0.0014	0.0004
	Sinusoidal	0.0010	0.0021	0.0016	0.0003
	Sigmoidal	0.0016	0.0114	0.0060	0.0025
Example 2	Variable	0.0009	0.0030	0.0014	0.0005
	Sinusoidal	0.0009	0.0045	0.0019	0.0010
	Sigmoidal	0.0097	0.0219	0.0132	0.0026
Example 3	Variable	0.0028	0.0128	0.0074	0.0029
	Sinusoidal	0.0026	0.0152	0.0064	0.0033
	Sigmoidal	0.0136	0.0544	0.0231	0.0092

Table 3: Learning Speed Statistics for Simulation Examples

	Network Type	*Min # Epochs*	*Max # Epochs*	*Mean # Epochs*	*Std. Deviation*
Example 1	Variable	683	786	734	33
	Sinusoidal	711	894	768	55
	Sigmoidal	1960	2258	2109	103
Example 2	Variable	634	801	723	45
	Sinusoidal	570	769	663	56
	Sigmoidal	1901	2227	2077	86
Example 3	Variable	667	906	770	57
	Sinusoidal	699	803	745	31
	Sigmoidal	1909	2399	2140	140

Table 4: Comparison in Performance

	RMS Ratio (Sig/Var)	*RMS Ratio (Sig/Sin)*	*Learning Ratio (Sig/Var)*	*Learning Ratio (Sig/Sin)*
Example 1	4.3	3.8	2.9	2.7
Example 2	9.4	6.9	2.9	3.1
Example 3	3.1	3.6	2.8	2.9

The above Tables illustrate the following key points:

- Due to the algorithmic preference for sinusoidal activation functions, variable networks and sinusoidal networks perform almost equivalently in both approximation error (2) and learning speed (3).
- Both the variable and sinusoidal networks outperform sigmoidal networks in approximation error for continuous functions by a factor ranging from 3 to 10 (4).
- Both the variable and sinusoidal networks learn approximately three times as fast as do the sigmoidal networks (4).

Since the performance of variable and sinusoidal networks is roughly equivalent, it sometimes is advantageous to use a sinusoidal network, rather than a variable network. By using only the one function type, a smaller pool of candidate units may be sufficient for training.

We have also performed a limited number of simulation runs comparing performance of variable and sigmoidal networks for higher-dimensional examples. From these simulations, we observed that the performance ratios reported in 4 remain roughly the same for multi input/output examples. This includes the dynamic system simulations in the following sections as well as in [18]. Since the results here so clearly suggest the use of variable or sinusoidal networks over sigmoidal networks, simulation results in the following sections are reported only for variable networks.

3.5 Comparison to Multilayer Feedforward Neural Networks

Here we compare the performance of variable cascade networks with multilayer feedforward networks (MLNNs). 5 summarizes simulation results for MLNNs, trained with the quickprop algorithm. In each case, we ran 5 separate trials. Although this is not an exhaustive study of every conceivable MLNN architecture, it does indicate faster learning and better continuous function approximation for variable cascade networks over multilayer networks.

Table 5: Performance Comparison between Multilayer Networks and the Cascade Architecture

Example	*MLNN Configuration*	*MLNN Free Parameters*	*Cascade Free Parameters*	*MLNN RMS*	*Variable RMS*	*MLNN Epochs*	*Variable Epochs*
1	1-25-1	76	77	0.0182	0.0014	2150	734
1	1-10-5-1	81	77	0.0086	0.0014	4200	734
2	1-25-1	76	77	0.0040	0.0014	5350	723
3	1-25-1	76	77	0.0594	0.0074	4750	771

Finally we point out that in [7], it was illustrated that one epoch of cascade learning generally requires far fewer computations than does one epoch of learning in a MLNN.

4 Dynamic System Identification

4.1 Mapping dynamic systems into static mappings

In general, a dynamic system may be expressed as a difference equation of the general form,

$$\bar{y}(k+1) = g(\bar{y}(k), \bar{y}(k-1), \ldots, \bar{y}(k-n), \bar{u}(k), \bar{u}(k-1), \ldots, \bar{u}(k-m)) \qquad \text{(Eq. 7)}$$

where *g(·)* is some arbitrary nonlinear function, $\bar{y}(k)$ is the output vector and $\bar{u}(k)$ is the input vector at time step *k*. The order of the dynamic system is given by the constants *n* and *m*, which may be infinite. Most neural networks are only capable of static input/output mapping, however. To overcome this problem, Narendra suggests providing a time history of data as input to the neural network [16][17]. Thus, *static* feedforward neural networks have the potential to approximate complex nonlinear mappings of *dynamic* systems for which no analytic model may exist. For example, Figure 18 illustrates how this is done for a SISO system of the form,

$$y(k+1) = f(y(k), y(k-1), y(k-2), u(k), u(k-1)) \qquad \text{(Eq. 8)}$$

In [17], special cases of (Eq. 7) are classified depending on whether part of the relationship in the equation is linear and on the extent of coupling between input and output states. Since the cascade architecture begins with direct linear connections between inputs and outputs, such classification is unnecessary here.

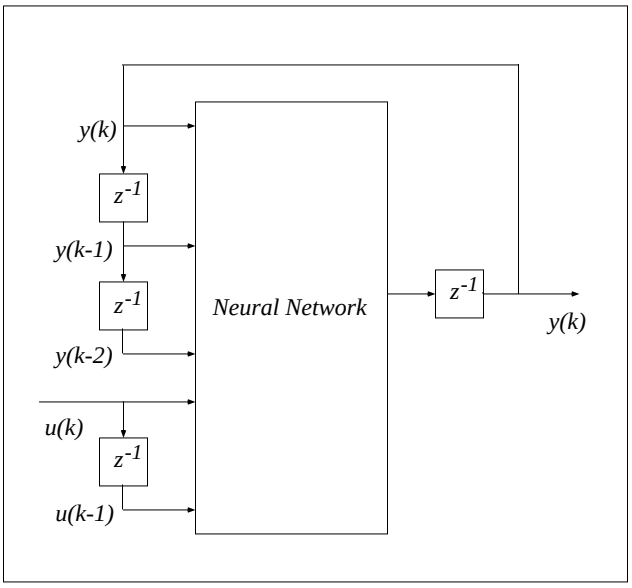

Figure 18: A dynamic system can be mapped onto a static feedforward neural network.

4.2 Control System Identification

Here, the dynamic system we wish the neural network to learn is a known, nonlinear control law for the classic inverted pendulum system. This is a traditional benchmark problem in control since the dynamics of the system are nonlinear and coupled, and the open-loop system is unstable. The dynamics of the system are governed by the following equations [9]:

$$\ddot{\theta} = \frac{3}{4l}(g\sin\theta - \ddot{x}\cos\theta) \quad \text{(Eq. 9)}$$

$$\ddot{x} = \frac{m\left(l\sin\theta\dot{\theta}^2 - \frac{3}{8}g\sin 2\theta\right) - f\dot{x} + u}{M + m\left(1 - \frac{3}{4}\cos^2\theta\right)} \quad \text{(Eq. 10)}$$

We use the following nonlinear control law as teacher to the neural network:

$$h_1 = \frac{3}{4l}g\sin\theta \quad \text{(Eq. 11)}$$

$$h_2 = \frac{3}{4l}\cos\theta \quad \text{(Eq. 12)}$$

$$f_1 = m\left(l\sin\theta\dot{\theta}^2 - \frac{3}{8}g\sin 2\theta\right) - f\dot{x} \quad \text{(Eq. 13)}$$

$$f_2 = M + m\left(1 - \frac{3}{4}\cos^2\theta\right) \quad \text{(Eq. 14)}$$

$$u = \frac{f_2}{h_2}[h_1 + k_1(\theta - \theta_d) + k_2\dot{\theta} + c_1(x - x_d) + c_2\dot{x}] - f_1 \quad \text{(Eq. 15)}$$

For the simulations, we used the following numeric values: $M = 1$ kg, $m = 0.1$ kg, $l = 1$ m, $f = 5$ kg/s, $g = 9.81$ m/s^2, $k_1 = 25$, $k_2 = 10$, $c_1 = 1$, $c_2 = 2.6$. Also we set $x_d = 0$ m, and $\theta_d = 0$ rad, which are the desired position of the cart and angle of the pendulum, respectively. For details on all the parameters see [9].

The system is simulated numerically using Euler's approximation method with a time step of $T = 0.02$ seconds. The neural network takes the following vector as input,

$$\{x(k-1), x(k), \theta(k), \theta(k-1)\} \quad \text{(Eq. 16)}$$

where $x(k-1)$ is the previous value of x, $x(k)$ the present value of x, $\theta(k-1)$ the previous value of θ, and $\theta(k)$ the present value of θ. For output, the network is trained to approximate the control law given by (Eq. 15). As training data, we generate 500 uniformly distributed random input/output vectors in the following range:

$$x(k) \in [-1 \text{ m}, 1 \text{ m}] \quad \text{(Eq. 17)}$$

$$\theta(k) \in [-0.5 \text{ rad}, 0.5 \text{ rad}] \quad \text{(Eq. 18)}$$

$$\{x(k) - x(k-1)\} \in [-0.04 \text{ m}, 0.04 \text{ m}] \quad \text{(Eq. 19)}$$

$$\{\theta(k) - \theta(k-1)\} \in [-0.02 \text{ rad}, 0.02 \text{ rad}] \qquad \text{(Eq. 20)}$$

After repeated trials, we found that as few as four hidden units in a variable or sinusoidal network were sufficient to model the controller, even for large initial values of θ. Below, we compare the performance of a trained neural network with four variable units to that of the actual control law. The initial conditions for the simulation are,

$$\{x, \dot{x}, \theta, \dot{\theta}\} = \{0, 0, 0.6 \text{ rad}, 0\} \qquad \text{(Eq. 21)}$$

Note that the initial condition for θ is outside the range of the training data. Figure 19 and Figure 20 show the system response for the nonlinear control law. Figure 21 and Figure 22 show the difference in response between the nonlinear controller and the neural network controller for both the cart position and pendulum angle. Considering that the neural network is presented with only an approximation of $\{\dot{x}, \dot{\theta}\}$, the learned behavior is remarkably close to the original control law. Furthermore, the neural network forms a stable controller for $-1.03 \text{ rad} < \theta_{initial} < 1.03 \text{ rad}$. Due to the discretization of time, the nonlinear control law becomes unstable for $|\theta_{initial}| > 1.10 \text{ rad}$.

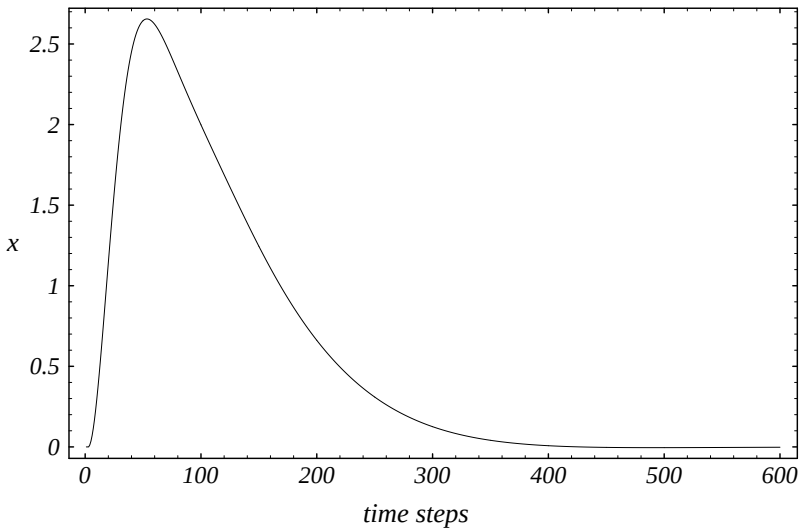

Figure 19: The cart position and velocity are driven outside the range of the training data for the network.

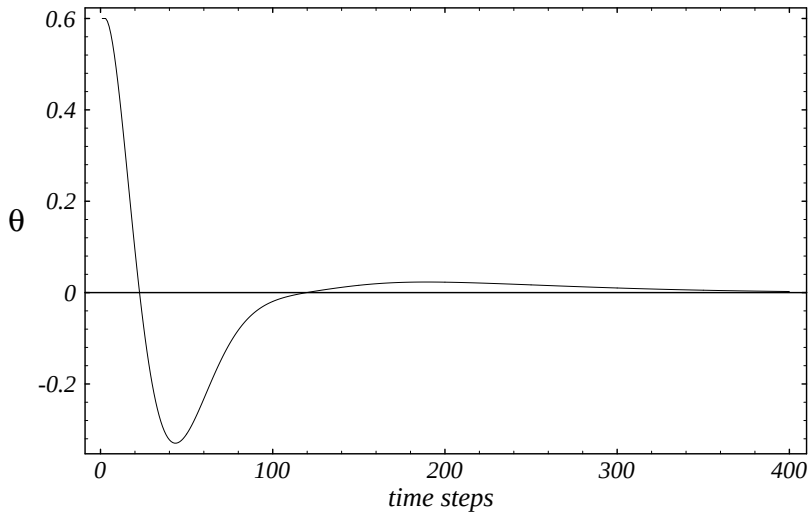

Figure 20: The initial pendulum angle starts outside the range of the training data for the network.

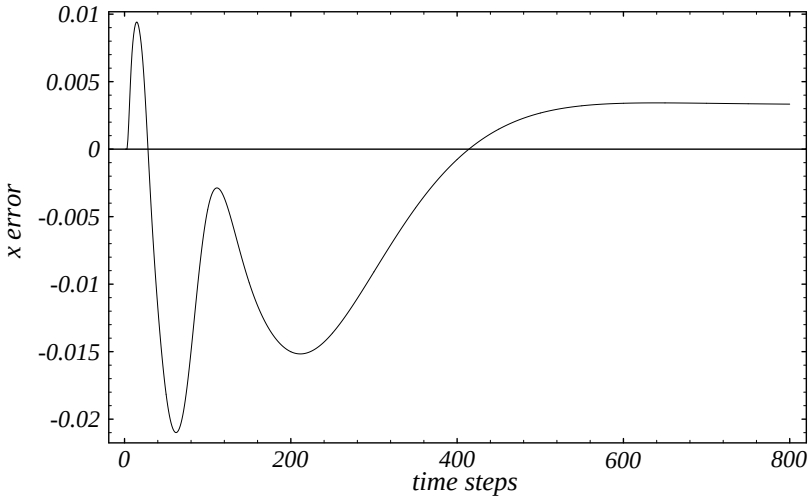

Figure 21: The neural network controller exhibits a small steady state error of approximately 3 mm.

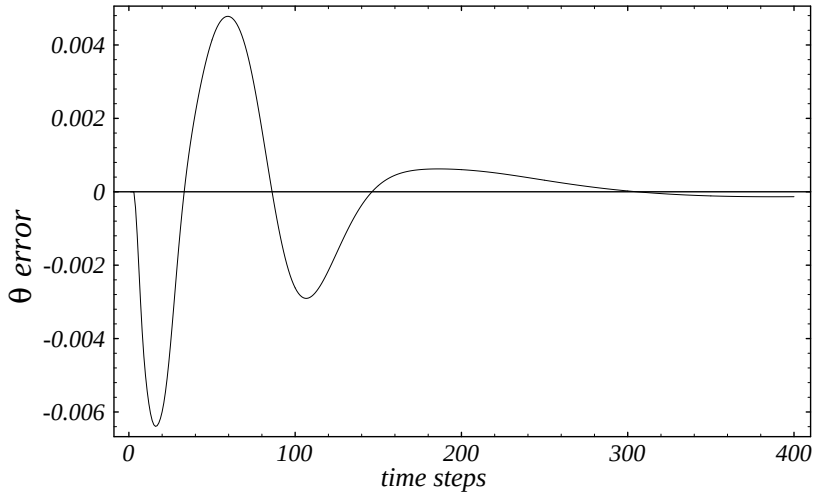

Figure 22: The pendulum angle converges to zero steady state error.

5 Identifying Human Control Strategy

In this section, we apply the results on continuous function approximation from the previous sections to the problem of identifying human control strategy. We again focus on the problem of controlling the inverted pendulum-cart system. Here, however, we replace the nonlinear control law used in the previous section with a human being as teacher for the neural network. The human teacher is asked to keep the pendulum from falling.

5.1 Experimental Setup

The setup for these experiments is illustrated below. A human subject is shown an inverted pendulum-cart system on a computer screen (Figure 23), and is able to control the horizontal force to be applied to the cart via the horizontal mouse position. The horizontal force that the user can apply is limited to $-64 \text{ N} \leq u \leq 64 \text{ N}$. Physical parameters for this cart-pendulum system are equivalent to those given in the previous section. The system state, as well as the control input provided by the human, are recorded at 100 Hz.

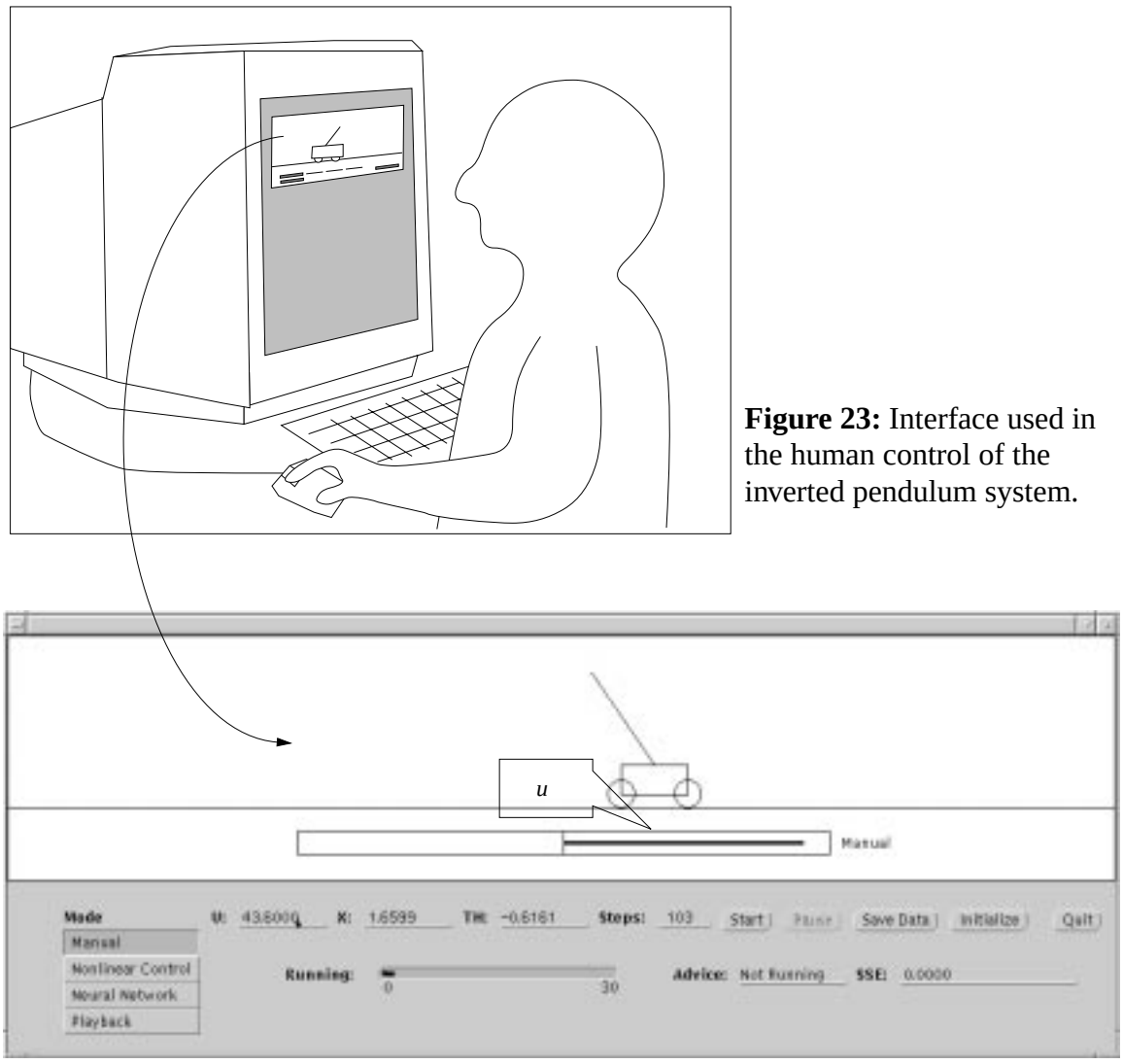

Figure 23: Interface used in the human control of the inverted pendulum system.

5.2 Identification Results

Case 1: After numerous failed attempts at keeping the pendulum from falling for any meaningful period of time, the first human subject successfully controls the system for 23 seconds or 2300 data points. Figure 24 shows the pendulum angle for that period of time, while Figure 25 shows the human-provided control input u. From this data, 750 randomly selected data points are selected to train the network. Another 750 randomly selected data points are used for cross validation.

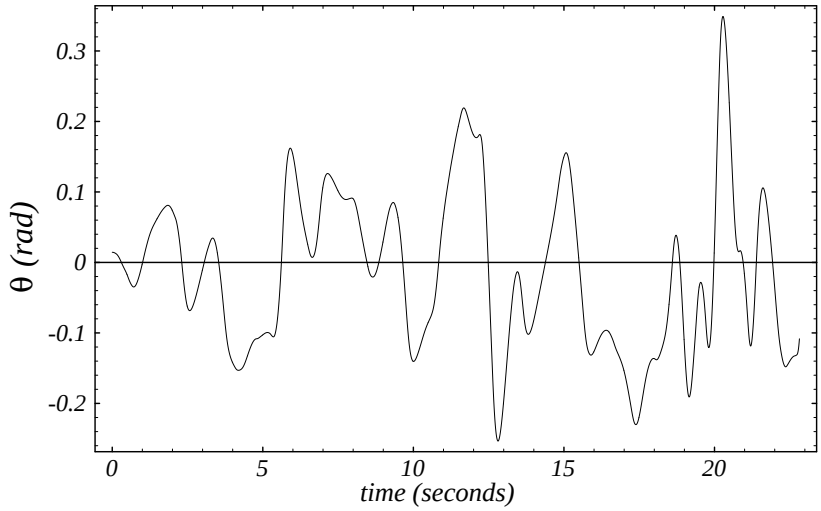

Figure 24: Pendulum angle for the complete successful training run of 23 seconds.

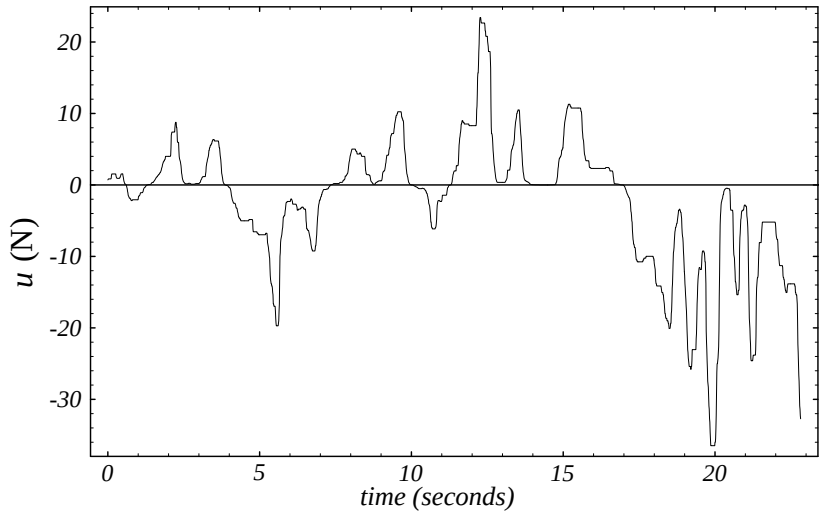

Figure 25: Control force for the complete successful training run of 23 seconds.

The neural network to be trained from this data takes six inputs, namely, the pas five values of θ, as well as the velocity of the cart $\dot{x}$,

$$\{\theta(k-4), \ldots, \theta(k-1), \theta(k), \dot{x}(k)\} \quad . \qquad \text{(Eq. 22)}$$

As output, the network is to generate the horizontal force to be applied to the cart for the next time step, $u(k+1)$. The resulting cascade network has twelve hidden units, with all the activation functions equal to one of the sinusoidal types. It was determined experimentally that this neural network controller is stable for $-0.92 \text{ rad} < \theta_{initial} < 0.98 \text{ rad}$.

The neural network control of the pendulum-cart system with initial conditions $\{x, \dot{x}, \theta, \dot{\theta}\} = \{0, 0, 0.2 \text{ rad}, 0\}$ is shown in Figure 26 and Figure 27 for 20 seconds. Figure 26 shows the pendulum angle θ, while Figure 27 shows the network-generated control force u.

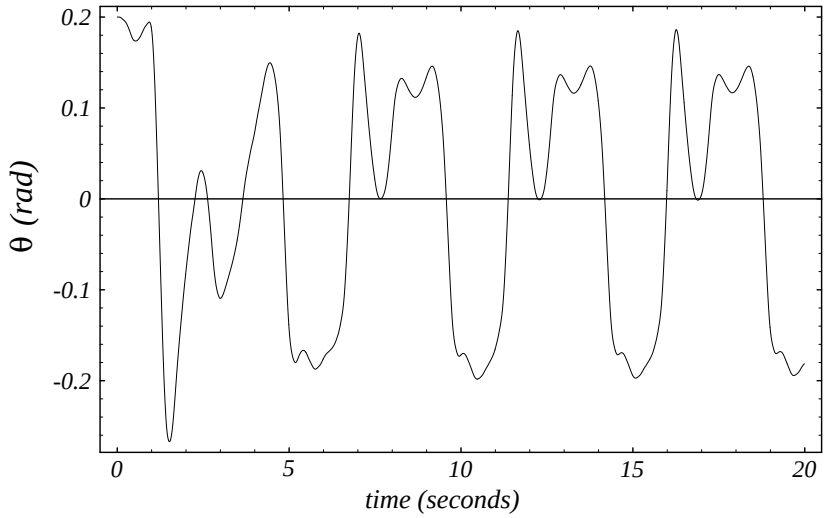

Figure 26: The neural network forms a stable, but nonconvergent controller.

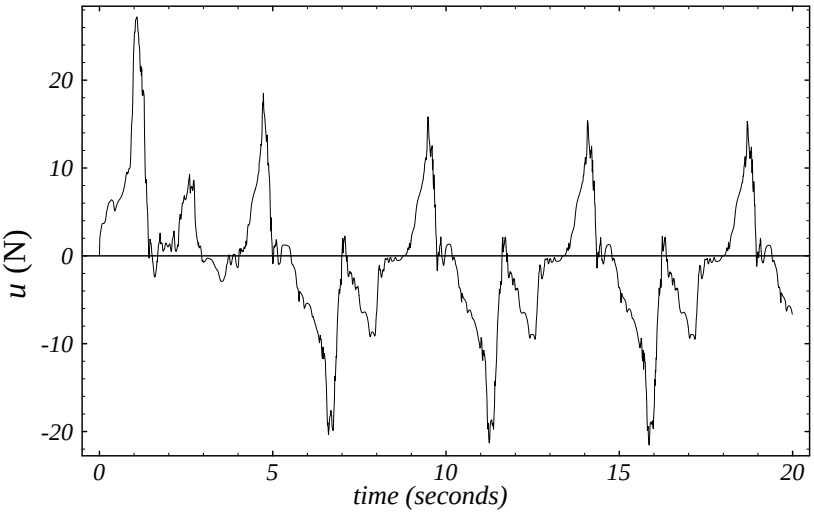

Figure 27: Control output generated by the neural network controller.

In Figure 28 below, 200 seconds of the pendulum trajectory are plotted in phase space. From this phase-space diagram, we observe that the trajectory, although not periodic, exhibits a definite pattern over a long period of time. In fact, the pendulum trajectory appears to form a possibly chaotic attractor. Over numerous simulation tests, the pendulum trajectory collapses to this attractor for all tested initial conditions within the region of stability.

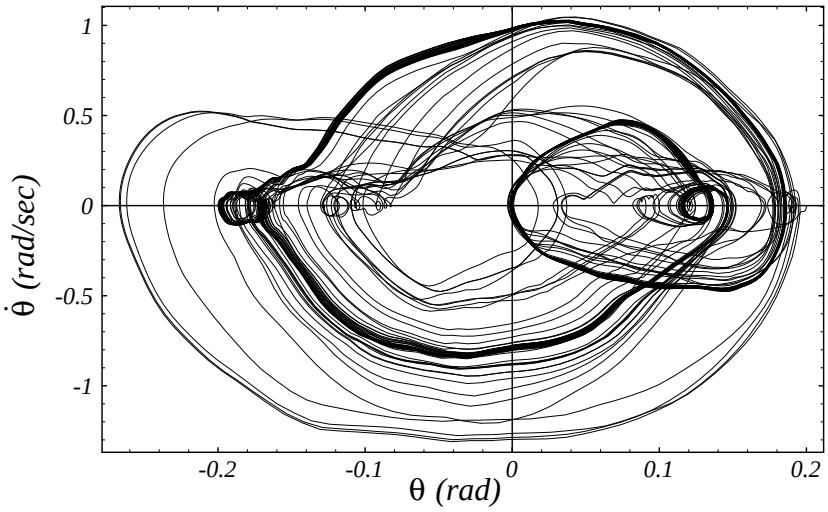

Figure 28: The pendulum trajectory appears to form a possibly chaotic attractor in phase space.

Case 2: For this case, a more skilled individual was asked to control the simulated inverted pendulum system. By "more skilled," we simply mean that the person could maintain stable control for a longer period of time. In fact, this subject successfully controls the system for approximately 60 seconds. Figure 29 shows the pendulum angle for 20 seconds of that successful run, while Figure 30 shows the human-provided control input u. From this data, 1000 randomly selected data points are selected to train the network. Another 1000 randomly selected data points are used for cross validation. This network has the same inputs and outputs as in *Case 1* above.

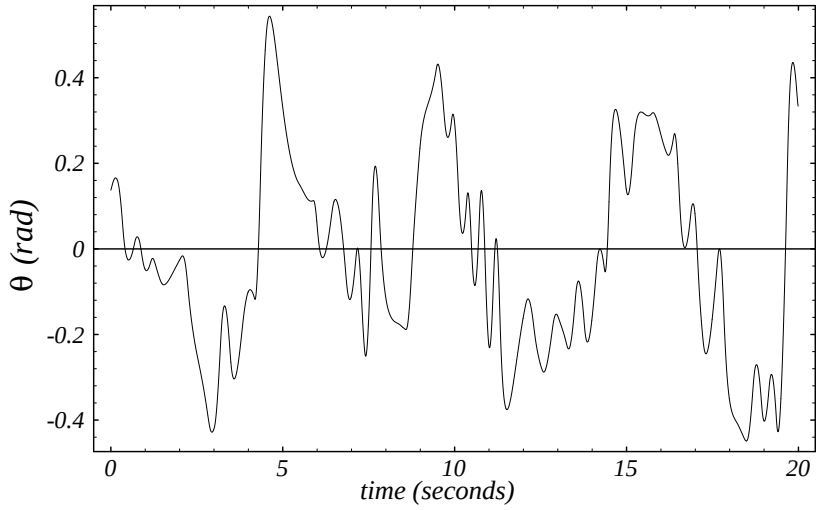

Figure 29: Part of the 60 second successful run for the second human controller.

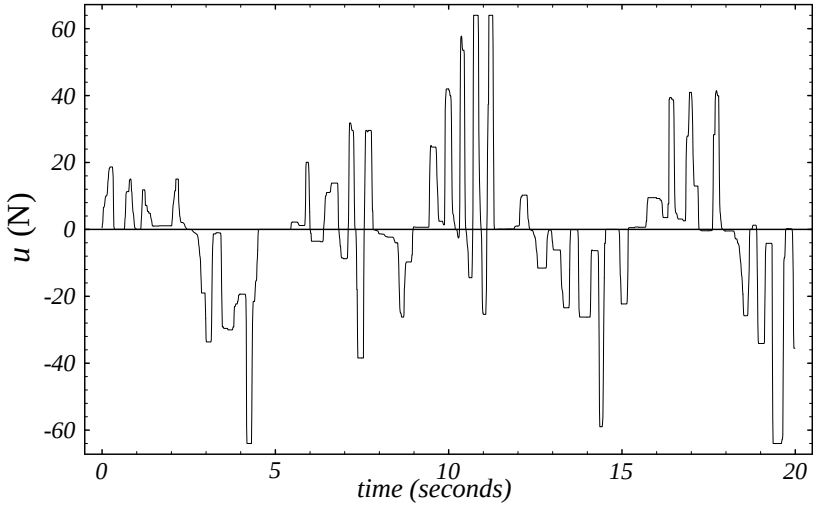

Figure 30: Control forces for the second human controller are higher than fore the first case.

The resulting cascade network has twelve hidden units, with all the activation functions equal to one of the sinusoidal types. Figure 31 and Figure 32 illustrate the neural network control of the pendulum-cart system with initial conditions $\{x, \dot{x}, \theta, \dot{\theta}\} = \{0, 0, 0.2 \text{ rad}, 0\}$ for 20 seconds. Figure 31 shows the pendulum angle θ, while Figure 32 shows the network-generated control force u.

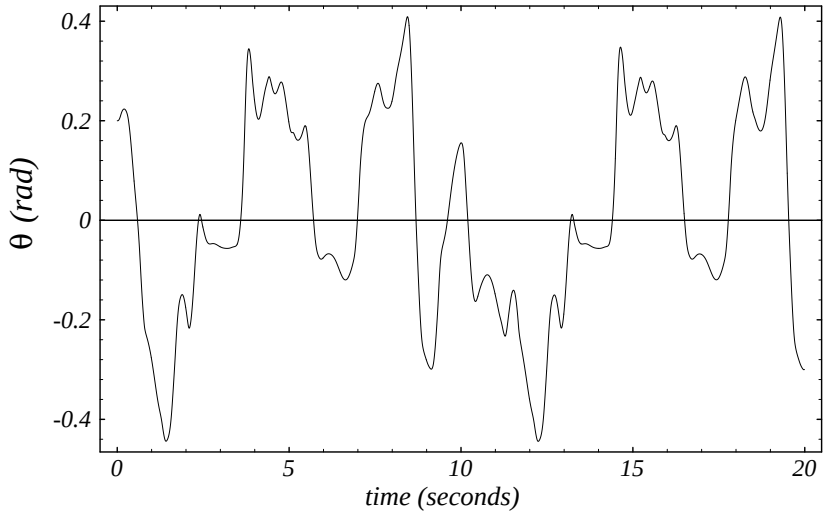

Figure 31: The neural network forms a stable controller.

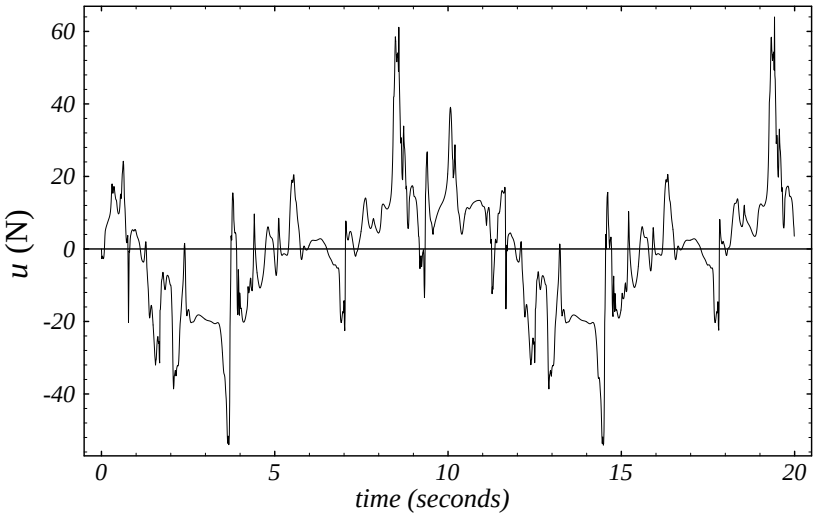

Figure 32: The neural network controller converges to encircle the origin in phase space.

The above pendulum trajectory again seems to form an attractor. As the simulation time is increased, however, there is a clear point where the characteristic trajectory fundamentally changes and collapses to a stable attractor. At what time this change occurs is dependent largely on the initial conditions. Different initial conditions lead to different times. For the trajectory in Figure 31, the change occurs at approximately 68 seconds. Figure 33 below plots the pendulum trajectory for the time interval around 68 seconds.

After the change in the characteristic trajectory, the pendulum trajectory converges to encircle the origin in an oscillating fashion. Thus, this controller, in fact, exhibits some convergent behavior. An examination of the weights in the network reveals several weights much larger than most other weights. The largest weight (in magnitude) is, for example, approximately 1201.

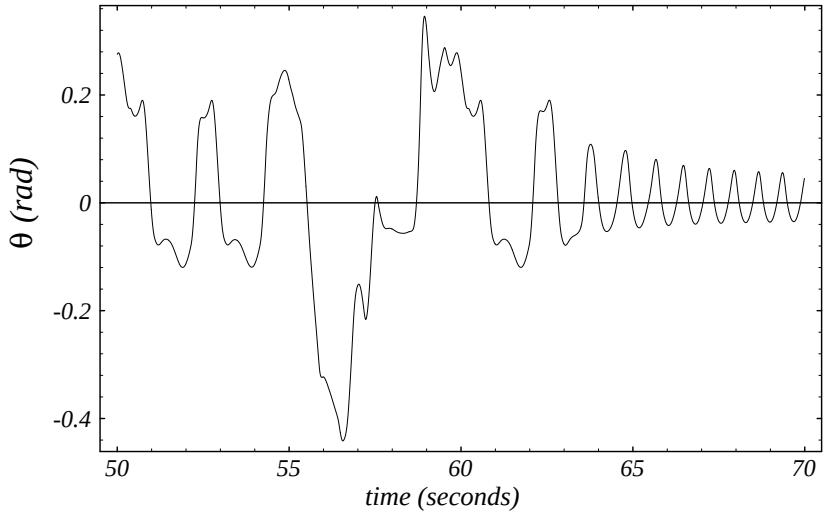

Figure 33: The neural network controller converges to encircle the origin in phase space.

In an effort to extract the important features from this neural network controller, small weights were zeroed (i.e. the corresponding connection between units was effectively cut). The resulting controller proves to be remarkably simple, and can be expressed by,

$$u(k+1) = -350[\theta(k)] + 1201[\theta(k-2)] - 925[\theta(k-4)] + 7.6\dot{x} \quad . \qquad \text{(Eq. 23)}$$

This controller is experimentally determined to be stable for $-1.04 \text{ rad} < \theta_{initial} < 1.05 \text{ rad}$. Figure 34 illustrates the controller response for initial conditions $\{x, \dot{x}, \theta, \dot{\theta}\} = \{0, 0, 0.6 \text{ rad}, 0\}$.

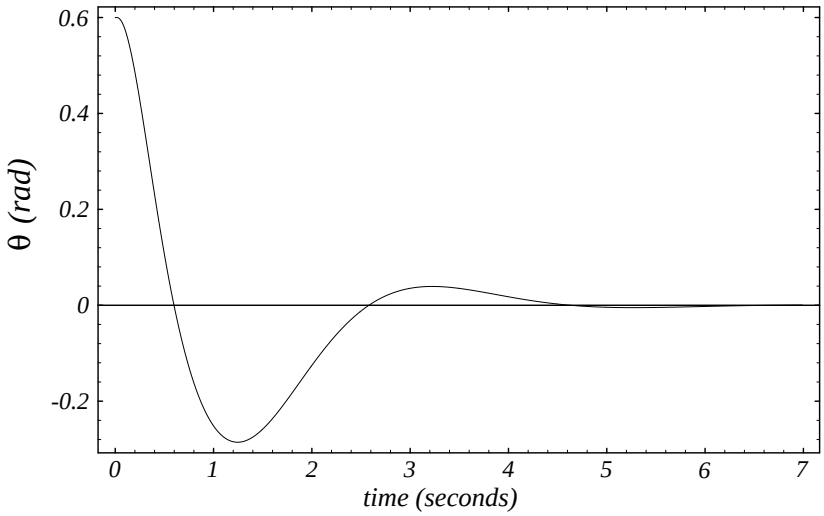

Figure 34: This controller was derived from training data provided by a human operator.

Thus, a convergent feedback controller has been abstracted from training data provided by a human operator. The first three terms in (Eq. 23) obviously combine to approximate the angular velocity of the pendulum.

5.3 Discussion & Future Work

Using the cascade two architecture, we are able to arrive at a compact and computationally efficient representation of the learned human control strategy. Although we have confined ourselves to off-line learning in this paper, the cascade two learning architecture can easily be extended to on-line, real-time learning. The primary difference between on-line and off-line learning is, of course, that, in off-line learning, the training data is static during learning, while in on-line learning, the training data is a dynamic, continuous stream of data. As such, deciding when a new hidden unit should be added during learning now must be based on a moving, rather than static, error measurement for real-time learning. When the dynamic training data is reasonably self-consistent, the learning will proceed the same in on-line learning as in off-line learning. We have found some problems, however, with convergence for on-line learning, and are currently addressing these problems.

In modeling human control strategy, many interesting issues remain to be explored and developed. First, we are curious whether or not learning has in fact modeled the important or essential aspects of the human control strategy. Deciding this is more difficult for modeling human control strategy than modeling a well-behaved, known control law. It is quite easy, for example, to evaluate the neural network controller, modeling the nonlinear control law of the

inverted pendulum system, in terms of standard error measurements for given test inputs. In the case of human modeling, however, no known target function, and hence no error measurements, exist to evaluate performance for task conditions not in the training set. Human training data is generally prone to errors and is stochastic in nature. Thus, we require different and new measurements of similarity between the human training data and the resulting neural network model. Such measurements can be statistical or heuristic in nature, but must be global, rather than local. Hidden Markov Models can be used to match training patterns to resulting network control patterns and may be one method to judge the fidelity of the network model with the inherent human model.

For the cases presented in this paper, there are clear similarities between the human training data and the resulting network performance. The control for case 1 is confined to a smaller range of pendulum angles, and is generally less turbulent than the control for case 2. This is clearly reflected in the network performances in Figure 26 and Figure 31. In fact, by observing the neural network control of the inverted pendulum on the graphical interface (Figure 23), both human subjects felt that the neural network accurately reproduced their control strategy; that is, it “looked like what [they were] doing” during the training phase. Any similarity measure that is ultimately developed should agree with these *ad hoc* observations made by the humans being modeled.

Another related concern with human modeling is not whether the model matches the human control strategy, but rather how good, objectively speaking, the model performs in carrying out a desired task. This, of course, depends on the skill of the human trainer. For the two cases in this paper, both human-modeled controllers have stability margins comparable to the discrete, nonlinear controller. The controller for case 1 keeps the pendulum within ± 0.2 rad, while the controller for case 2 keeps the pendulum within ± 0.4 rad. Thus, it may appear that the first controller is better in terms of performance than the second controller. The second controller, however, can be simplified, to reveal an elegant structure, that is not only stable, but also convergent. Moreover, this controller rivals the partitioned, nonlinear control law in performance. No such simplification was possible for case 1. Therefore, the better controller in both performance and simplicity resulted from the better skilled individual, as one might expect.

This, however, raises another interesting issue, namely, how a model based on human training data can or should be used. In the first case, it appears that the best achievable result, given the training data, is the consequent trained model, requiring no post-processing. In the second case, however, a tremendously improved result was achieved by merely zeroing relatively

small weights. Identification of a human control strategy, therefore is one matter; how the resulting model can then best be utilized for control is quite another matter. We are exploring generalizable methods of abstracting the underlying structure of a human performance model, so as to improve the usability and control performance of the human model.

Another interesting application of modeling human control strategy lies in teaching the good control of a skilled individual to another person, whose control is less skilled. If we can, in fact, generate a good model of skilled human control strategy, others might then learn, not from the skilled person directly, but rather from the model of the skilled person. Essentially, the less-skilled individual is asked to train his/her neural network (i.e. brain), by attempting to emulate the skilled model through feedback data. Potentially, a single expert could train far more people through this indirect approach, rather than being directly involved in the training of others on a case-by-case basis. Human-to-human transfer of skill may then also facilitate human-to-robot skill transfer, and vis versa.

These are but some of the issues and problems, related to modeling human control strategy, that we are currently working on, and will be addressed in the future. We believe that they are well worth solving. Ultimately, an approach to solving complex task problems in robotics that does not first look towards humans as a guide neglects the single biggest source of solved problems in visual processing, manipulation, and mobility.

6 Conclusion

Learning control from humans by example is an important concept for making robots and machines more intelligent. Neural networks are well suited to generate the complex nonlinear mapping of the human control process, which maps sensory inputs to control action outputs. We have presented encouraging results for nonlinear continuous function mapping and dynamic system identification by utilizing a new neural network architecture. We have demonstrated that the cascade network architecture with variable activation functions is well suited for efficiently mapping continuous nonlinear functions. We have also demonstrated that the method allows a neural network to learn both a known nonlinear, coupled control law, as well as unknown nonlinear human control strategy. Finally, we discuss some potentially exciting areas for future research from the results in human control strategy identification.

7 Acknowledgments

We would like to thank Scott Fahlman for his advice in using the cascade two learning architecture. Also, we would like to thank Scott Fahlman, Michael Kingsley, David C. Lambert for the use of their neural network simulation software.

8 References

[1] Antsaklis, P. J., Guest Editor, Special Issue on Neural Networks in Control Systems, *IEEE Control System Magazine*, Vol. 10, No. 3, pp. 3-87, 1990.

[2] Antsaklis, P. J., Guest Editor, Special Issue on Neural Networks in Control Systems, *IEEE Control System Magazine*, Vol. 12, No. 2, pp. 8-57, 1992.

[3] Asada, H., and Liu, S., "Transfer of Human Skills to Neural Net Robot Controllers," *Proc. IEEE Int. Conf. on Robotics and Automation*, Vol. 3, pp. 2442-2447, 1991.

[4] Cybenko, G., "Approximation by Superposition of a Sigmoidal Function," *Mathematics of Control, Signals, and Systems*, Vol. 2, No. 4, pp. 303-314, 1989.

[5] Fahlman, S. E., "An Empirical Study of Learning Speed in Back-Propagation Networks," Technical Report, CMU-CS-TR-88-162, Carnegie Mellon University, 1988.

[6] Fahlman, S.E., and Boyan, J.A., "The Cascade Two Learning Architecture," Technical Report (forthcoming), CMU-CS-94-100, Carnegie Mellon University, 1994.

[7] Fahlman, S.E., and Lebiere, C., "The Cascade-Correlation Learning Algorithm," Technical Report, CMU-CS-90-100, Carnegie Mellon University, 1990.

[8] Funahashi, K., "On the Approximate Realization of Continuous Mappings by Neural Networks," *Neural Net.*, Vol. 2, No. 3, pp. 183-192, 1989.

[9] Guez, A., Selinsky, J., "A Trainable Neuromorphic Controller," *Jour. of Robotic Sys.*, Vol 5., No. 4, pp. 363-388, 1988.

[10] Hornik, K., Stinchcombe, M., and White, H., "Universal Approximation of an Unknown Mapping and Its Derivatives Using Multilayer Feedforward Networks," *Neural Net.*, Vol. 3, No. 5, pp. 551-560, 1990.

[11] Hunt, K. J., *et. al.*, "Neural Networks for Control Systems - A Survey," *Automatica*, Vol. 28, No. 6, pp. 1083-1112, 1992.

[12] Kurkova, V., "Kolmogorov's Theorem and Multilayer Neural Networks," *Neural Net.*, Vol. 5, No. 3, pp. 501-506, 1992.

[13] Lee, C. C., “Fuzzy Logic in Control Systems: Fuzzy Logic Controller - Part I,” *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. 20, No. 2, pp. 404-418, 1990.

[14] Lee, C. C., “Fuzzy Logic in Control Systems: Fuzzy Logic Controller - Part II,” *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. 20, No. 2, pp. 419-435, 1990.

[15] Miller, W. T., Sutton, R. S., and Werbos, P. I., Editors, “Neural Networks For Control,” MIT Press, 1990.

[16] Narendra, K. S., “Adaptive Control of Dynamical Systems Using Neural Networks,” *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*, White D. A., and Sofge D. A., ed., pp. 141-184, 1992.

[17] Narendra, K. S., Parthasarathy, K., “Identification and Control of Dynamical Systems Using Neural Networks,” *IEEE Trans. on Neural Networks*, Vol. 1, No. 1, pp. 4-27, 1990.

[18] Nechyba, M., and Xu, Y., “Neural Network Approach to Control System Identification with Variable Activation Functions,” P*roc. IEEE Int. Symp. Intelligent Control*, pp. 358-363, 1994.

[19] Pomerleau, D. A., “Neural Network Perception for Mobile Robot Guidance,” Ph.D. Thesis, School of Computer Science, Carnegie Mellon University, 1992.

[20] Yang, J., Xu, Y., and Chen, C., “Hidden-Markov Model-Based Learning Controller,” *Proc. IEEE Int. Symp. Intelligent Control*, pp. 39-44, 1994.