Chapter 4

Neural Network Substrates for Control Instincts

Chapter Contents

4.1 Biological Neural Networks and Their Role in Control 107
4.1.1 Neurons and Neural Networks 107
4.1.2 Example: Instinctual Neural Control Functions in Simple Organisms . . 109
4.2 Multilayer Perceptrons 111
4.2.1 The Neuron 112
4.2.2 Feedforward Network of Neurons 114
4.3 Design Example: Multilayer Perceptron for Tanker Ship Steering . 116
4.3.1 Tanker Ship Model and Heading Regulation 116
4.3.2 Construction of a Multilayer Perceptron for Ship Steering 121
4.3.3 Multilayer Perceptron Stimulus-Response Characteristics 125
4.3.4 Behavior of the Ship Controlled by the Multilayer Perceptron 126
4.4 Radial Basis Function Neural Networks 131
4.5 Design Example: Radial Basis Function Neural Network for Ship Steering 133
4.5.1 A Radial Basis Function Neural Network for Ship Steering 133
4.5.2 Stimulus-Response Characteristics 138
4.5.3 Behavior of the Ship Controlled by the Radial Basis Function Neural Network 139
4.6 Stability Analysis 141
4.6.1 Differential Equations and Equilibria 141
4.6.2 Stability Definitions 144
4.6.3 Lyapunov's Direct Method for Stability Analysis 145
4.6.4 Stability of Discrete Time Systems 146
4.6.5 Example: Stable Instinctual Neural Control 147
4.7 Hierarchical Neural Networks 148
4.7.1 Example: Marine Mollusc 149
4.7.2 Hierarchical Neural Structures 149
4.8 Exercises and Design Problems 149

We first provide examples of how biological neural networks help to implement instinctual control functionalities in some simple organisms. Then, departing somewhat from biology, we introduce two types of neural network models that ignore many details of real neurons and their interconnections (e.g., voltage spikes and dynamics) to produce "firing rate models" with specific forms for "tuning curves." In this chapter we also ignore learning and evolution and thereby only model a special type of control instinct with limited functional capabilities (e.g., the functions and parameters of our models will not change over time). Also, we only model functionality of the neural network, and not the many other sensory and actuation functions needed for an organism to achieve control.

Next, we explain how our neuron models can be viewed as building blocks (sets of tuning curves) for creating a neural network that can implement a complex input-output mapping. To do this, we show how to take the mappings implemented by neurons and build by hand ("design") a controller for a specific engineering application. In biological systems, over long time periods, this "design" is a task of evolution; over short time periods such as a life time, it is the task of learning. Aside from ignoring learning and evolution, this design approach will at the same time represent an even more significant departure from biology as we ignore whether it has a biological basis (e.g., in development) and whether the constructed neural networks bear any similarity to those in any biological system. We only concern ourselves with constructing input-output mappings that will lead to the performance objectives being met. Biology provides building blocks with basic functionalities, and we pay no respect to whether we use these building blocks as biology would.

For specific engineering applications, we show how to simulate the neural network controlling the plant and evaluate whether the closed-loop system meets performance objectives. This will provide insights into neural network stimulus-response characteristics and their effect on closed-loop behavior. Moreover, it will serve as an introduction on how to evaluate control systems in simulation. Ultimately, however, certain difficulties in the design process, and the need for controllers with more sophisticated functionalities that will be highlighted in the simulations, will motivate the need to study learning in Part III and evolutionary methods in Part IV that automate the construction of neural networks.

4.1 Biological Neural Networks and Their Role in Control

First, we briefly outline some basics of how biological neural networks operate, specifically in controlling functions of a few simple organisms.

4.1.1 Neurons and Neural Networks

An invertebrate motor neuron was shown in Figure 2.3. There are, however, many different types of neurons, with, for example, the cell body at different

There are massively interconnected networks of neurons of many forms in organisms; engineering models often use the invertebrate motor neuron connected in special topologies.

locations. The "multipolar" configuration shown in Figure 2.3(a) is the one that is often modeled and then used in control engineering (and indeed, it underlies many other studies of neural networks in other engineering areas). Each neuron is composed of "dendrites" which allow for connections to the cell body, an "axon" which allows for connections to other neurons, and the connection points are called the "synapses." Generally, the cell body performs a type of summing and thresholding operation on signals obtained via the dendrites and provides an electrical signal (actually it is typically a sequence of pulses that are often called "spikes") that travels along the axon to other neurons. When such a signal is transmitted on the axon, the neuron is said to "fire." The inputs to the neuron on the dendrites can be "excitatory" (having a tendency to cause the neuron to fire) or "inhibitory" (having the tendency to restrict the firing of the neuron).

It is via the massive neural interconnectivity that complex reasoning and intelligence emerges.

The human brain is composed of a large, massively interconnected biological network of about 10^{11} neurons, each of which may have as many as 10^3 or 10^4 connections to other neurons (for a total of up to $100,000$ miles of neuron connections). These neurons dynamically interact with each other, change their properties over time (e.g., via learning), and even grow new connections to each other (e.g., during fetal development), to act as a sophisticated bioelectrical "computer" of sorts. While the interconnection of neurons in humans is extremely sophisticated (see Figure 4.1), here we only consider simple interconnections. For instance, the three neurons in Figure 4.2 are connected in a "feedforward" fashion (i.e., without connecting an axon of one neuron back to another neuron that has a path to that neuron), since this is a common interconnection strategy used in engineering applications.

Processing characteristics of individual neurons and network interconnections, and hence topology of the network, change via learning.

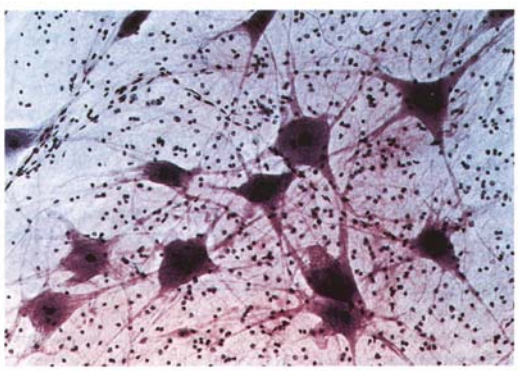

Figure 4.1: Network of motor neurons in the spinal cord, photograph taken through a microscope (figure taken from [223], © 1991, 1994, and 1999 by Worth Publishers Inc., and used with permission).

There are a number of neurons in our body and in other organisms that are "hard-wired" in the sense that their properties are fixed in a specific manner

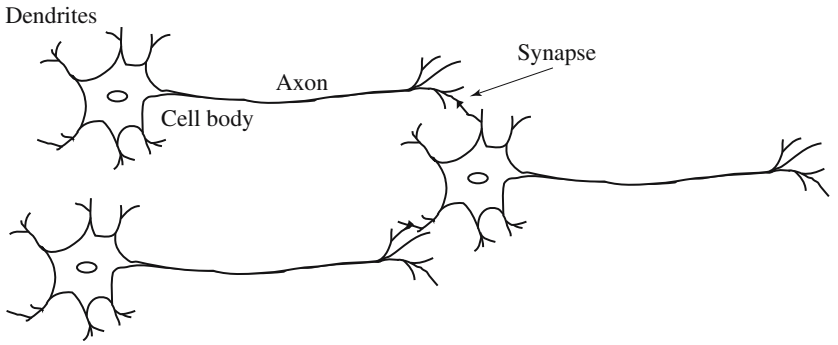

Figure 4.2: Three connected neurons, a simple biological neural network.

such that they have no ability to learn. For example, certain motor reflexes are implemented this way, and the functions that they implement are then sometimes said to be "instinctual" (i.e., they do not need to be learned). While there are a variety of functions in a human (or other organisms) that are instinctual or automatic, a perhaps more interesting case is when a neural network develops and neurons have an ability to learn. For instance, when humans are born they have very low neural network connectivity in their brain. As they develop and learn, the brain forms interconnections between different neurons and the resulting network defines the functional properties of the brain (yes, the environment does affect the actual connection structure in our brain). Moreover, to memorize information it is said that the interconnections between the neurons are modified and then fixed so that the information can later be recalled. This learning capability will be more fully investigated in Part III; here, the focus is on "instinctual" control functions, examples of which are given next.

4.1.2 Example: Instinctual Neural Control Functions in Simple Organisms

"Command systems" of neurons are used in biological systems for a variety of tasks, such as control of motion, locomotion, digestion, etc. Many animal behaviors, such as walking or swimming, result from a network of neurons called a "central pattern generator" that produces a pattern of signals that results in a rhythmic contraction and relaxation of muscles. More generally, instinctive responses are sometimes called "fixed action patterns" that are evoked by a "sign stimulus." Such behaviors can be quite complex, but are thought of as being rigidly evoked by a certain type of stimulus (i.e., the animal does not learn these responses, or forget them).

In this section we will show how in one organism, neurons can control movement and in another organism, how the neural network can respond to stimuli to produce movement. The goal here is simply to show neurons and neural networks "at work" in acting as controllers in biological systems.

Neurons That Control Swimming in a *Clione*

A simple neural "circuit" (interconnection of neurons) that can produce alternating contraction and relaxation in two different muscles that move "wing-like" structures called *parapodia* in the *Clione* (a small marine mollusc), is shown in Figure 4.3. There are two neurons for moving each wing, one for "upswing" and the other for "downswing." Each neuron has an inhibitory effect on the other so that when one is active, the other is not (i.e., it is inhibited by the other). The signals on the right show the basic electrical pattern between the neurons where when the voltage in the upswing neuron spikes (and actuates a muscle for moving the wing up), it inhibits the other neuron. However, after the upswing spike has decreased, the downswing neuron voltage increases and then spikes, which signals the muscle to move the wing down. The firing of one neuron to actuate the upswing inhibits the firing of the downswing neuron, and vice versa. This is called "reciprocal inhibition" and it is the key feature that allows the command system of neurons to generate rhythmic movement.

Neural networks with only a few neurons can control movements in simple organisms that are very useful for survival (e.g., locomotion for foraging or predator avoidance).

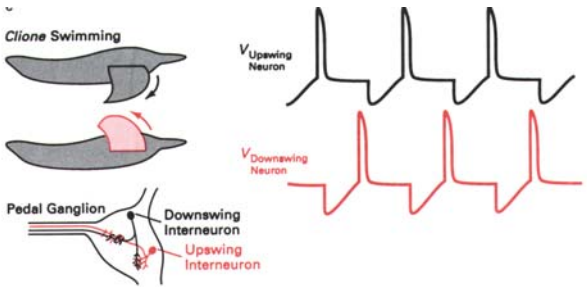

Figure 4.3: Command system of neurons (neural controller) for swimming in a *Clione* (figure taken from [312], © Oxford University Press, reproduced by permission).

Neuron Stimulus-Response Actions to Achieve Control in a Swimming Leech

The pulse-type voltage patterns ("spikes") in Figures 4.3 and 4.4 are typical for neurons; however, most engineering models do not represent neuron behavior to this level of detail.

The medicinal leech *Hirudo medicinalis* swims by making undulating motions with its body, somewhat like a snake or some fish (see Figure 4.4). The movements result from alternating contraction and relaxation of muscles that are located in the body wall of the leech. When it swims, there are rhythmic bursts from a central pattern generator that produce a "wave" of contraction that travels from the front to the rear of the leech. Reciprocal inhibition is used to produce the rhythmic motion in the leech, just as it was in the *clione* discussed in the last subsection.

The leech will start swimming if there is a brief strong mechanical stimulus applied to the body of the animal as shown in Figure 4.4. There, a sensory neuron detects the stimulus and starts firing (in the figure the "firing" is characterized by the spikes in the signal voltage of the sensory neuron). This sets off

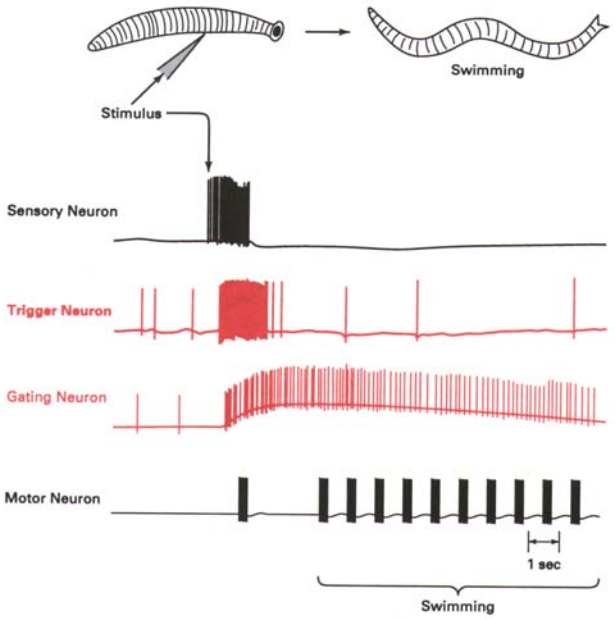

Figure 4.4: Neuron signaling connecting stimulus to swimming response in a medicinal leech *Hirudo medicinalis* (figure taken from [312], © Oxford University Press, reproduced by permission).

a "trigger neuron" that in turn starts a "gating neuron" to fire, and the firing of that gating neuron persists even when the stimulus is removed. When the gating neuron is active (i.e., when it fires), it sustains the rhythmic activity of the central pattern generator. The gating neuron makes a motor neuron active at regular one second intervals and these signal the muscle for swimming.

4.2 Multilayer Perceptrons

Next, we will explain how we model the physiological system of the neural network. It must be emphasized that the models here are not meant to be precise models of parts of a biological brain or neurons in any other organism. Essentially, they are "firing rate models" since they do not model voltage spikes, but have outputs that are thought of as being proportional to the frequency of the spikes. Moreover, they are "static" since they do not include, for example, dynamic systems to represent that currents or voltages in a neuron cannot change instantaneously. First, we consider a multilayer perceptron which is a feedforward neural network (e.g., it does not use past values of its outputs to compute its current output). It is composed of an interconnection of basic neuron processing units.

4.2.1 The Neuron

For a single neuron, suppose that we use x_i, $i = 1, 2, \ldots, n$, to denote its inputs and suppose that it has a single output y. Figure 4.5 shows the neuron. Such a neuron first forms a weighted sum of the inputs

$$\bar{x} = \left(\sum_{i=1}^{n} w_i x_i \right) + b$$

where w_i are the interconnection "weights" and b is the "bias" for the neuron. The signal x_i is the input to the i^{th} dendrite and $w_i > 0$ represents an excitatory connection with larger w_i values representing dendrites that "amplify" their input signals more. Conversely, $w_i < 0$ represents an inhibitory input. The signal $\bar{x}$ represents a signal in the biological neuron that represents the combined effects of all the inputs from the dendrites.

The stimulus-response characteristics of a neuron can be changed by adjusting the weights w_i, bias b, or by using different types of activation functions f.

The processing that the neuron performs on this $\bar{x}$ signal is represented with an "activation function." This activation function is represented with a function f, and the output that it computes is

$$y = f(\bar{x}) = f\left(\left(\sum_{i=1}^{n} w_i x_i \right) + b \right) \tag{4.1}$$

Basically, the neuron model represents the biological neuron that "fires" (turns on and passes an electrical signal down the axon so that it can go to other neurons as shown in Figure 4.2) when its inputs are significantly excited (i.e., $\bar{x}$ is big enough). Normally, Equation 4.1 is represented as shown in Figure 4.5.

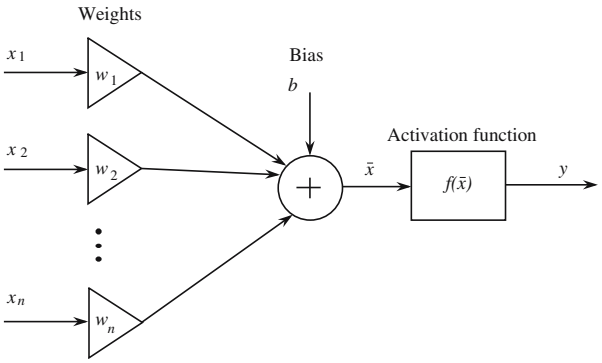

Figure 4.5: Single neuron model.

The manner in which the neuron fires is defined by the activation function f. There are many ways to define the activation function:

- *Threshold function:* For this type of activation function, we have

$$f(\bar{x}) = \begin{cases} 1 & \text{if } \bar{x} \geq 0 \\ 0 & \text{if } \bar{x} < 0 \end{cases}$$

so that once the input signal $\bar{x}$ is above zero the neuron turns on (see Figure 4.6).

- *Linear function:* For this type of activation function, we simply have

$$f(\bar{x}) = \bar{x}$$

and we think of the neuron being on when $f(\bar{x}) > 0$ and off when $f(\bar{x}) \leq 0$ (see Figure 4.6).

- *Logistic function:* For this type of activation function, which is a type of "sigmoid function," we have

$$f(\bar{x}) = \frac{1}{1 + \exp(-\bar{x})} \tag{4.2}$$

so that the input signal $\bar{x}$ continuously turns on the neuron an increasing amount as the input increases as shown in Figure 4.6 (note that for the logistic function $f(0) = 0.5 \neq 0$ but $f(0) - 0.5 = 0$ where 0.5 can be modeled by another bias so that you can think of the logistic function as "turning on" in a similar way to how the other functions in Figure 4.6 turn on).

- *Hyperbolic tangent function:* There are many functions that take on a shape that is sigmoidal. For instance, one that is often used in neural networks is the hyperbolic tangent function

$$f(\bar{x}) = \tanh(\bar{x}) = \frac{1 - \exp(-2\bar{x})}{1 + \exp(-2\bar{x})}$$

which is shown in Figure 4.6.

Equation (4.1), with one of the above activation functions, represents the "computations" made by one neuron in a neural network. Notice that the input-output characteristics of a neuron in a multilayer perceptron are quite different from the biological neurons discussed in the last section. Along with the assumption that the weights, a bias, and a summing operation represent part of what happens in the dendrites and cell body, the activation function output (and hence firing of the neuron) is not represented as a voltage spike that travels down the axon, or a sequence of such spikes (such as in Figure 4.4) that might be frequency modulated by the overall activation level of the neuron (e.g., have higher frequency spikes for greater activation levels as is sometimes found in a biological neuron). Essentially, a larger input to the activation function here (for a sigmoid function) simply turns the neuron on to a greater extent; the neuron here is a very simple (abstract) model of the behavior of *some* biological neurons called a "firing rate model." It is interesting to note, however, that the specific shapes for the above activation functions (and some others) have been experimentally demonstrated for firing rate models of real neurons, so there is some biological justification for the form or the model we use here. The specific

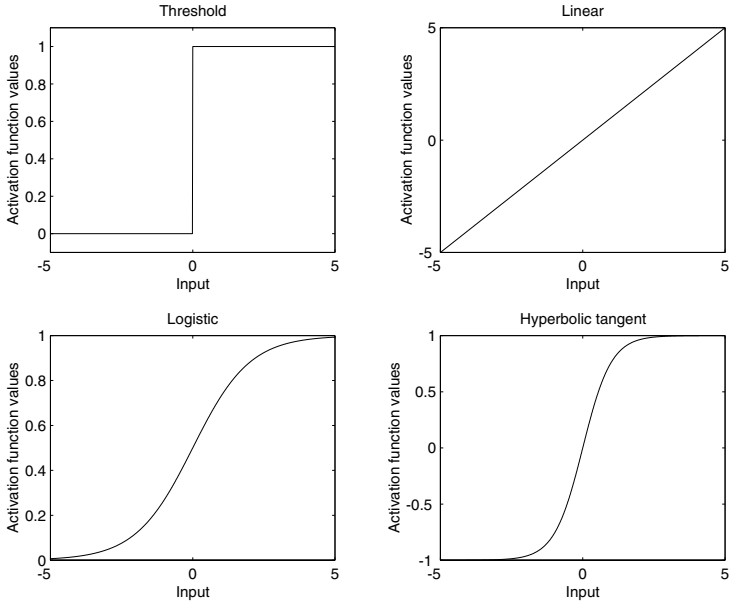

Figure 4.6: Activation functions for neurons.

shapes of the mappings produced by individual neurons are sometimes called "tuning curves" or "tuning functions" (they show how the neuron is "tuned" to a stimulus by showing for a whole range of stimulus inputs what the firing rate output of the neuron will be). Some neurons have tuning curves in the shape of sigmoids so that for some stimuli they are not on but as the stimulus changes, the neuron starts firing at a high rate above some threshold, if the slope of the sigmoid near the threshold is steep. Other neurons to be modeled in Section 4.4 have tuning curves in the shape of Gaussian functions so that they are "on" the most for a specific range of stimuli. See the "For Further Study" section at the end of this part.

Next, we define how we interconnect neurons to form a neural network—in particular, the feedforward multilayer perceptron.

4.2.2 Feedforward Network of Neurons

The basic structure for the multilayer perceptron is shown in Figure 4.7. There, the circles represent the neurons (weights, bias, and activation function) and the lines represent the connections between the inputs and neurons, and between the neurons in one layer and those in the next layer. This is a three-layer perceptron since there are three stages of neural processing between the inputs and outputs. The layer connected to the output is called the "output layer," and all the other ones are called "hidden" layers since they do not connect to the output.

The multilayer perceptron has inputs x_i, $i = 1, 2, \ldots, n$, and outputs y_j, $j = 1, 2, \ldots, m$. The number of neurons in the first hidden layer (see Figure 4.7) is n_1. In the second hidden layer there are n_2 neurons, and in the output layer there are m neurons. Hence, in an N layer perceptron there are n_i neurons in the i^{th} hidden layer, $i = 1, 2, \ldots, N-1$.

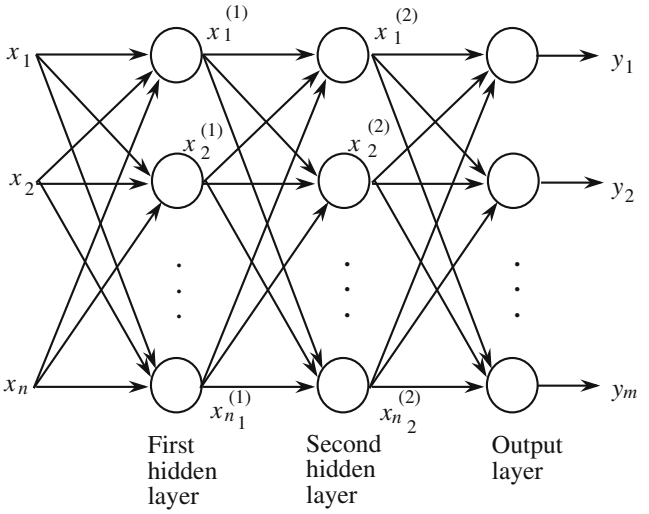

Figure 4.7: Multilayer perceptron model.

The neurons in the first layer of the multilayer perceptron perform computations, and the outputs of these neurons are given by

$$x_j^{(1)} = f_j^{(1)} \left(\left(\sum_{i=1}^{n} w_{ij}^{(1)} x_i \right) + b_j^{(1)} \right)$$

with $j = 1, 2, \ldots, n_1$. The neurons in the second layer of the multilayer perceptron perform computations, and the outputs of these neurons are given by

$$x_j^{(2)} = f_j^{(2)} \left(\left(\sum_{i=1}^{n_1} w_{ij}^{(2)} x_i^{(1)} \right) + b_j^{(2)} \right)$$

with $j = 1, 2, \ldots, n_2$. The neurons in the third layer of the multilayer perceptron perform computations, and the outputs of these neurons are given by

$$y_j = f_j \left(\left(\sum_{i=1}^{n_2} w_{ij} x_i^{(2)} \right) + b_j \right)$$

with $j = 1, 2, \ldots, m$.

The parameters (scalar real numbers) $w_{ij}^{(1)}$ are called the weights of the first hidden layer. The $w_{ij}^{(2)}$ are called the weights of the second hidden layer. The

w_{ij} are called the weights of the output layer. The parameters $b_j^{(1)}$ are called the biases of the first hidden layer. The parameters $b_j^{(2)}$ are called the biases of the second hidden layer, and the b_j are the biases of the output layer. The functions f_j (for the output layer), $f_j^{(2)}$ (for the second hidden layer), and $f_j^{(1)}$ (for the first hidden layer) represent the activation functions. The activation functions can be different for each neuron in the multilayer perceptron (e.g., the first layer could have one type of sigmoid, while the next two layers could have different sigmoid functions or threshold functions).

The simulus-response characteristics of a neural network can be changed via neuron parameters or their interconnections.

For convenience, we sometimes use

$$y = F_{mlp}(x, \theta)$$

to denote the multilayer perceptron where θ is a parameter vector that holds all the tunable weights and biases of the multilayer perceptron.

This completes the definition of the multilayer perceptron. Next, we will show how a multilayer perceptron's stimulus-response characteristics can be designed so that it can be used to regulate the heading of a ship.

4.3 Design Example: Multilayer Perceptron for Tanker Ship Steering

Here, we show how a neural network can be used to steer a ship that is traveling on the ocean. To do this, we first define the ship model and heading regulation problem. Next, we define the neural network, design its stimulus-response characteristics, and then evaluate how it performs in its ship steering task.

4.3.1 Tanker Ship Model and Heading Regulation

Our tanker ship heading regulation problem is shown in Figure 4.8. Here, the ship is moving forward in the indicated x direction at a nominal speed u, ψ denotes the heading angle (in radians), and δ is the rudder input (in radians). We will use ψ_r to denote the desired ship heading that is specified, for instance, by the captain (or route planner). The goal is to develop a control system that will ensure that ψ tracks ψ_r.

It is very important to achieve good heading regulation for ships since this reduces consumption of fuel. Steering performance can be affected by a variety of variables. It is known that the ship can travel at different speeds and this affects how the ship is steered (the rudder becomes less effective at very low speeds), that in general the ship weighs different on different trips (and heavy ships turn slower), that wind can be encountered on some trips and when wind hits the side of the tanker this can affect heading regulation, that water currents can affect steering, and that the sensor for ship steering provides a noisy measurement. Also, the rudder will only move between ± 80 degrees and this affects our ability to steer the ship.

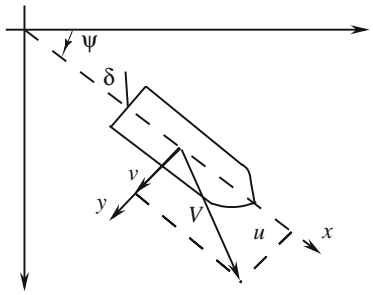

Figure 4.8: Tanker ship steering problem.

Ship Model

In order to study the behavior of the system, we will simulate it on a digital computer. To do this, we need to develop a computer program that is based on a nonlinear model of the ship; we will develop this model next. Often, ship dynamics are obtained by applying Newton's laws of motion to the ship. For very large ships, the motion in the vertical plane may be neglected since the "bobbing" or "bouncing" effects of the ship are small for large vessels. The motion of the ship is generally described by a coordinate system that is fixed to the ship [30] as shown in Figure 4.8.

A simple model of the ship's motion is given by

$$\dddot{\psi}(t) + \left(\frac{1}{\tau_1} + \frac{1}{\tau_2}\right)\ddot{\psi}(t) + \left(\frac{1}{\tau_1\tau_2}\right)\dot{\psi}(t) = \frac{K}{\tau_1\tau_2}\left(\tau_3\dot{\delta}(t) + \delta(t)\right) \tag{4.3}$$

where ψ is the heading of the ship and δ is the rudder angle. Assuming zero initial conditions, we can write Equation (4.3) as

$$\frac{\psi(s)}{\delta(s)} = \frac{K(s\tau_3 + 1)}{s(s\tau_1 + 1)(s\tau_2 + 1)} \tag{4.4}$$

where K, τ_1, τ_2, and τ_3 are parameters that are a function of the ship's constant forward velocity u and its length l. In particular,

$$\begin{aligned} K &= K_0\left(\frac{u}{l}\right) \\ \tau_i &= \tau_{i0}\left(\frac{l}{u}\right) \qquad i = 1, 2, 3 \end{aligned}$$

where we assume that for a tanker ship under "ballast" conditions (a very heavy ship), $K_0 = 5.88$, $\tau_{10} = -16.91$, $\tau_{20} = 0.45$, $\tau_{30} = 1.43$, and $l = 350$ meters [30]. For "full" conditions (a lighter ship), $K_0 = 0.83$, $\tau_{10} = -2.88$, $\tau_{20} = 0.38$, $\tau_{30} = 1.07$. If we do not say otherwise, we will simulate the tanker ship under ballast conditions. Also, we will assume that nominally the tanker ship is traveling in the x direction at a velocity of $u = 5$ m/s.

In normal steering, a ship often makes only small deviations from a straight-line path. Therefore, the model in Equation (4.3) was obtained by linearizing the equations of motion around the zero rudder angle ($\delta = 0$). As a result, the rudder angle should not exceed approximately 5 degrees, otherwise the model will be inaccurate. For our purposes, we need a model suited for rudder angles that are larger than 5 degrees; hence, we use the model proposed in [51]. This extended model is given by

$$\dddot{\psi}(t) + \left(\frac{1}{\tau_1} + \frac{1}{\tau_2}\right)\ddot{\psi}(t) + \left(\frac{1}{\tau_1\ \tau_2}\right)H(\dot{\psi}(t)) = \frac{K}{\tau_1\ \tau_2}\left(\tau_3\ \dot{\delta}(t) + \delta(t)\right) \quad (4.5)$$

where $H(\dot{\psi})$ is a nonlinear function of $\dot{\psi}(t)$. The function $H(\dot{\psi})$ can be found from the relationship between δ and $\dot{\psi}$ in steady state such that $\dddot{\psi} = \ddot{\psi} = \dot{\delta} = 0$. An experiment known as the "spiral test" has shown that $H(\dot{\psi})$ can be approximated by

$$H(\dot{\psi}) = \bar{a}\dot{\psi}^3 + \bar{b}\dot{\psi}$$

where $\bar{a}$ and $\bar{b}$ are real-valued constants and $\bar{a}$ is always positive. We choose the values $\bar{a} = \bar{b} = 1$. Also, we assume that the maximum deviation of the rudder angle is ± 80 degrees (or $80\pi/180$ radians).

Simulation of Nonlinear Systems

The ship model is nonlinear; hence, in order to simulate its behavior on a digital computer we need to discuss how to simulate nonlinear systems. In this subsection we give a brief overview of how to simulate general nonlinear systems. In the next subsection, we will return to the ship example and show how to develop a simulation for its behavior.

Suppose that the system to be simulated can be represented by the ordinary differential equation

$$\begin{aligned} \dot{x}(t) &= f(x(t), r(t), t) \\ y &= g(x(t), r(t), t) \end{aligned} \quad (4.6)$$

where $x = [x_1, x_2, \ldots, x_n]^\top$ is a state vector, $f = [f_1, f_2, \ldots, f_n]^\top$ is a vector of nonlinear functions, g is a nonlinear function that maps the states and reference input to the output of the system, and $x(0)$ is the initial state. Note that f and g are, in general, time-varying functions due to the explicit dependence on the time variable t. To simulate a nonlinear system, we will assume that the nonlinear ordinary differential equations are put into the form in Equation (4.6).

Euler's Method: Now, to simulate Equation (4.6), we could simply use Euler's method to approximate the derivative $\dot{x}$ in Equation (4.6) as

$$\begin{aligned} \frac{x(kh+h) - x(kh)}{h} &= f(x(kh), r(kh), kh) \\ y &= g(x(kh), r(kh), kh) \end{aligned} \quad (4.7)$$

Here, h is a parameter that is referred to as the "integration step size." Notice that any element of the vector

$$\frac{x(kh+h)-x(kh)}{h}$$

is simply an approximation of the slope of the corresponding element in the time varying vector $x(t)$ at $t = kh$ (i.e., an approximation of the derivative). For small values of h, the approximation will be accurate provided that all the functions and variables are continuous. Equation (4.7) can be rewritten as

$$\begin{aligned} x(kh+h) &= x(kh) + hf(x(kh), r(kh), kh) \\ y &= g(x(kh), r(kh), kh) \end{aligned}$$

for $k = 0, 1, 2, \ldots$. The value of the vector $x(0)$ is the initial condition and is assumed to be given. Simulation of the nonlinear system proceeds recursively by computing $x(h)$, $x(2h)$, $x(3h)$, and so on, to generate the response of the system for the reference input $r(kh)$.

Note that by choosing h small, we are trying to simulate the *continuous-time* nonlinear system. If we want to simulate the way that a digital control system would be implemented on a computer in the laboratory, we can simulate a controller that only samples its inputs every T seconds (T is not the same as h; it is the "sampling interval" for the computer in the laboratory) and only updates its control outputs every T seconds (and it would hold them constant in between). Normally, you would choose $T = \alpha h$ where $\alpha > 0$ is some positive integer. In this way, we simulate the plant as a continuous-time system that interacts with a controller that is implemented on a digital computer.

The Runge-Kutta Method: While Euler's method is easy to understand and implement in code, sometimes to get good accuracy the value of h must be chosen to be very small. Most often, to get good simulation accuracy, more sophisticated methods are used, such as the Runge-Kutta method with adaptive step size or predictor-corrector methods. In the fourth-order Runge-Kutta method, we begin with Equation (4.6) and a given $x(0)$ and let

$$x(kh+h) = x(kh) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{4.8}$$

where the four vectors

$$\begin{aligned} k_1 &= hf(x(kh), r(kh), kh) \\ k_2 &= hf\left(x(kh) + \frac{k_1}{2}, r\left(kh + \frac{h}{2}\right), kh + \frac{h}{2}\right) \\ k_3 &= hf\left(x(kh) + \frac{k_2}{2}, r\left(kh + \frac{h}{2}\right), kh + \frac{h}{2}\right) \\ k_4 &= hf\left(x(kh) + k_3, r(kh+h), kh+h\right) \end{aligned}$$

These extra calculations are used to achieve a better accuracy than the Euler method. We see that the Runge-Kutta method is very easy to use; it simply involves computing the four vectors k_1 to k_4, and plugging them into Equation (4.8). Suppose that you write a computer subroutine to compute the output of a fuzzy controller given its inputs (in some cases these inputs could include a state of the closed-loop system). In this case, to calculate the four vectors, k_1 to k_4, you will need to use the subroutine four times, once for the calculation of each of the vectors, and this can increase the computational complexity of the simulation. The complexity is reduced, however, if you can simulate the fuzzy controller as if it were implemented on a digital computer in the laboratory with a sampling interval of $T = \alpha h$ (see the discussion above). Also, sometimes $r(kh)$ is used in place of $r(kh + h/2)$ and $r(kh + h)$ in the above equations; this can be a reasonable approximation if r is constant most of the time and f and g are not time-varying functions (i.e., they do not have t as one of their arguments).

Generally, if the Runge-Kutta method has a small enough value of h, it is sufficiently accurate for the simulation of most control systems (and if an adaptive step size method is used, then even more accuracy can be obtained if it is needed). For more details on numerical simulation of nonlinear differential equations, see [332, 217, 508].

Simulating the Ship and a Digital Controller

Next, we need to convert the n^{th}-order nonlinear ordinary differential equations representing the ship to n first-order ordinary differential equations; for convenience, let

$$a = \left(\frac{1}{\tau_1} + \frac{1}{\tau_2}\right)$$

$$b = \left(\frac{1}{\tau_1 \tau_2}\right)$$

$$c = \frac{K\tau_3}{\tau_1 \tau_2}$$

and

$$d = \frac{K}{\tau_1 \tau_2}$$

We would like the model in the form

$$\begin{aligned} \dot{x}(t) &= f(x(t), \delta(t)) \\ y(t) &= g(x(t), \delta(t)) \end{aligned}$$

where $x(t) = [x_1(t), x_2(t), x_3(t)]^\top$ and $f = [f_1, f_2, f_3]^\top$ for use in a nonlinear simulation program. We need to choose $\dot{x}_i$ so that f_i depends only on x_i and δ for $i = 1, 2, 3$. We have

$$\dddot{\psi}(t) = -a\ddot{\psi}(t) - bH(\dot{\psi}(t)) + c\dot{\delta}(t) + d\delta(t) \tag{4.9}$$

Choose

$$\dot{x}_3(t) = \dddot{\psi}(t) - c\dot{\delta}(t)$$

so that f_3 will not depend on $c\dot{\delta}(t)$ and

$$x_3(t) = \ddot{\psi}(t) - c\delta(t)$$

Choose $\dot{x}_2(t) = \ddot{\psi}(t)$ so that $x_2(t) = \dot{\psi}(t)$. Finally, choose $x_1(t) = \psi$. This gives us

$$\begin{aligned}
\dot{x}_1(t) &= x_2(t) = f_1(x(t), \delta(t)) \\
\dot{x}_2(t) &= x_3(t) + c\delta(t) = f_2(x(t), \delta(t)) \\
\dot{x}_3(t) &= -a\ddot{\psi}(t) - bH(\dot{\psi}(t)) + d\delta(t)
\end{aligned}$$

But, $\ddot{\psi}(t) = x_3(t) + c\delta(t)$, $\dot{\psi}(t) = x_2(t)$, and $H(x_2) = x_2^3(t) + x_2(t)$ so

$$\dot{x}_3(t) = -a\left(x_3(t) + c\delta(t)\right) - b\left(x_2^3(t) + x_2(t)\right) + d\delta(t) = f_3(x(t), \delta(t))$$

Also, we have $\psi = g(x, \psi_r) = x_1$. This provides the proper equations for the simulation. Next, suppose that the initial conditions are $\psi(0) = \dot{\psi}(0) = \ddot{\psi}(0) = 0$. This implies that $x_1(0) = x_2(0) = 0$ and $x_3(0) = \ddot{\psi}(0) - c\delta(0)$ or $x_3(0) = -c\delta(0)$.

For the ship steering problem, we let the integration step size be $h = 1$ sec. and $\alpha = 10$ so that $T = \alpha h = 10$ sec. (i.e., the controller is implemented on a digital computer with a sampling period of $T = 10$ sec. so that a new plant input is calculated every 10 sec. and applied to the rudder). We will use this same approach for all the simulations for the tanker ship in this book.

4.3.2 Construction of a Multilayer Perceptron for Ship Steering

Here, we construct a simple multilayer perceptron for steering the ship. To do this, we must first choose the controller inputs and outputs. We will assume that the only input for steering is the rudder angle δ (we will not consider the speed u to be an input; it will be fixed). Hence, δ is the only output of the multilayer perceptron. The choice of inputs to the multilayer perceptron depends on what variables of the tanker can be sensed. It is assumed that the reference input ψ_r is given (e.g., if you use a neurophysiological view you may think of it as being provided by the frontal lobes in the brain that perform route planning functions). To keep things simple we will assume that we can only sense the ship heading ψ (if you took a neurophysiological view, this would then have to be provided by visual processing or some other sensory input that is then connected to the neurons that make the steering decisions). With these choices, the control system block diagram for the ship steering problem is shown in Figure 4.9.

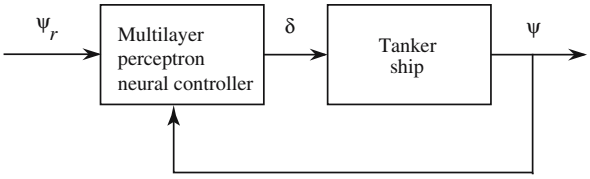

Figure 4.9: Control system for using a multilayer perceptron for tanker ship steering.

Structure Choice and The First Hidden Layer

From Figure 4.9, we see that the multilayer perceptron is a mapping from ψ_r and ψ to δ, the input to the ship. Suppose we denote the multilayer perceptron as $\delta = F_{mlp}(\psi_r, \psi)$ (note that for convenience we omit the arguments indicating the dependence on the parameters of the network). Our objective is to specify the mapping $F_{mlp}(\psi_r, \psi)$ by picking the number of layers of neurons, the number of neurons in each layer, and the specific weights, biases, and activation functions for all the neurons (from a neurophysiological view, it is evolution that specifies this mapping and we will simply construct one that we hypothesize evolution could have constructed). To do this, we will simply show one possible choice for the multilayer perceptron and explain some of the reasoning behind its construction. Consider Figure 4.10. There, we use a four layer perceptron with both linear and logistic sigmoidal activation functions. First, consider the first hidden layer. For this we choose $w_{11}^{(1)} = 1$, $w_{21}^{(1)} = -1$, and $b_1^{(1)} = 0$. Hence, we see that the output of the first layer is the heading error

A neural network can be designed to compare signals for use in "decision-making" (e.g., comparing a desired value to a sensed one).

$$e = \psi_r - \psi$$

To a control engineer this may seem to be an odd approach to implement a simple summing junction to provide the heading error; however, it is interesting that a neuron can provide a method to compare two signals, something that is certainly of fundamental importance in making control decisions for tracking and regulation.

Choosing Weights and Biases: Building Nonlinearities with Smooth Step Functions

Next, we explain how to pick the weights and biases for the remaining layers. To do this, view the perceptron in Figure 4.10 as having two "paths" of processing from the signal e that is the output of the first hidden layer to the output δ. Imagine that you remove the path on the bottom and first focus on constructing the path on the top. We will think of the top path as being used to regulate the ship heading when

$$e = \psi_r - \psi \geq 0$$

In this case, we want to have a *negative* rudder input. To see this, consider Figure 4.8 where you can see that a positive rudder input results in a *decrease*

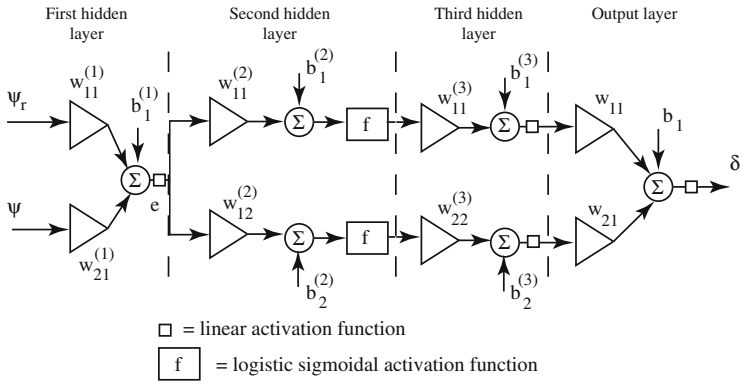

Figure 4.10: A multilayer perceptron for tanker ship steering.

in the heading ψ and a negative rudder input results in an *increase* in the heading ψ. Hence, if $\psi_r - \psi \geq 0$, we have $\psi_r \geq \psi$ so that we want to increase the size of ψ, so we use a negative rudder input δ. Next, note that for larger values of $|e| = |\psi_r - \psi|$, we will generally want larger values of δ since larger heading errors generally require larger rudder inputs to reduce them quickly. How do we choose the weights and biases in the top path to implement this type of control action?

Neural network construction can be viewed as "building" stimulus-response characteristics from basic neuron building blocks that are deformable via their parameters (here, we build functions from "smooth steps").

To specify values for the weights and biases, we think of each neuron as providing a type of "smooth switching" (a smooth step function as shown in Figure 4.6, e.g., for the logistic function) and tune the weights and biases to adjust these and build nonlinear control mappings. Note that when only the top path is considered, we have

$$\delta = w_{11}\left(\frac{w_{11}^{(3)}}{1+\exp(-\bar{x})} + b_1^{(3)}\right) + b_1$$

where

$$\bar{x} = b_1^{(2)} + w_{11}^{(2)} e$$

The parameters in these equations affect the shape of the nonlinearity from e to δ in the following manner:

- b_1, $b_1^{(3)}$: Shift the mapping up and down.
- w_{11}, $w_{11}^{(3)}$: Scale the vertical axis.
- $b_1^{(2)}$: Shifts the smooth step (logistic function) horizontally, with $b_1^{(2)} > 0$ shifting it to the *left*.

- $w_{11}^{(2)}$: Scale the horizontal axis (you may think of this as a type of gain for the function, at least locally).

Using these ideas, choose

$$\begin{aligned} b_1 &= b_1^{(3)} = 0 \\ w_{11} &= 1 \\ w_{11}^{(3)} &= -\frac{80\pi}{180} \\ b_1^{(2)} &= -\frac{200\pi}{180} \\ w_{11}^{(2)} &= 10 \end{aligned}$$

With these choices we get the nonlinear mapping shown in the top plot of Figure 4.11. The general shape of the function is appropriate to use as a controller for $e > 0$ since it provides negative rudder input values for positive values of error, and it provides a type of proportionality between the size of e and the size of δ. Notice that the choice of $w_{11}^{(3)}$ results in the perceptron providing a maximum negative rudder deflection of -80 degrees. The choice of $b_1^{(2)}$ simply shifts the function to the right, so that the value of the function near $e = 0$ provides $\delta \approx 0$. The value of $w_{11}^{(2)}$ affects the slope of the function as $e > 0$ increases in size; if $w_{11}^{(2)}$ were chosen to be larger, then it would reach the maximum negative value of -80 degrees quicker as the size of e increases. This completes the construction of the perceptron for the top path, which is dedicated to control for the case where $e \geq 0$.

Next, consider the bottom path of Figure 4.10 (imagine disconnecting the top path) which is dedicated to the case $e < 0$. Using the same ideas for the choice of the parameters above, select

$$\begin{aligned} b_2^{(3)} &= \frac{80\pi}{180} \\ w_{21} &= 1 \\ w_{22}^{(3)} &= -\frac{80\pi}{180} \\ b_2^{(2)} &= \frac{200\pi}{180} \\ w_{12}^{(2)} &= 10 \end{aligned}$$

The resulting nonlinearity implemented by the bottom path is shown in the middle plot of Figure 4.11. First, note that its general shape is appropriate to use as a controller for the case where $e < 0$; as the size of e increases in the negative direction, increasingly positive values of a rudder input δ are provided to try to decrease the value of ψ to the given ψ_r. The values of $w_{22}^{(3)}$, $b_2^{(2)}$, and $w_{12}^{(2)}$ were chosen in a similar way as the corresponding values for the top path were chosen. The value of $b_2^{(3)}$ was chosen to shift the nonlinearity up

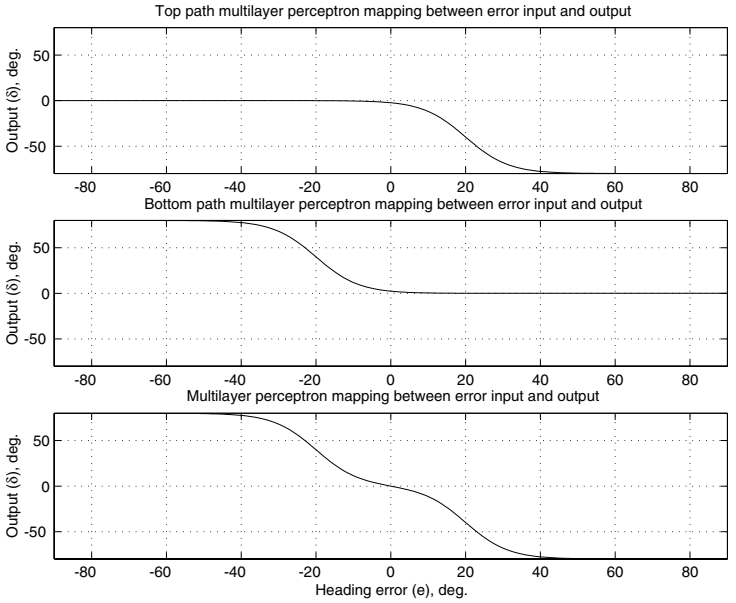

Figure 4.11: Multilayer perceptron mappings, top plot is for the top path of the perceptron from e to δ, middle plot is for the bottom path of the perceptron from e to δ, bottom plot is for the entire perceptron from e to δ.

by 80 degrees. The choice for w_{21} completes the specification of the output layer, which simply sums the functions generated by the top and bottom paths, and results in the overall mapping from e to δ shown in the bottom plot of Figure 4.11 (i.e., when both the top and the bottom paths in Figure 4.10 are used). Notice that due to the symmetry in our choices, $e = 0$ implies that $\delta = 0$ so that if the ship is going in the right direction, the rudder does not try to correct for the heading direction. This completes the construction of the multilayer perceptron for regulating the ship heading.

4.3.3 Multilayer Perceptron Stimulus-Response Characteristics

The multilayer perceptron construction procedure in the last subsection showed how to construct the controller

$$\delta = F_{mlp}(\psi_r, \psi)$$

Given a stimulus represented by particular values of ψ_r and ψ, the perceptron will react and provide a response δ according to how the function $F_{mlp}(\psi_r, \psi)$ is shaped. In this way the $F_{mlp}(\psi_r, \psi)$ nonlinearity implements a "control surface," which in this case has the shape shown in Figure 4.12.

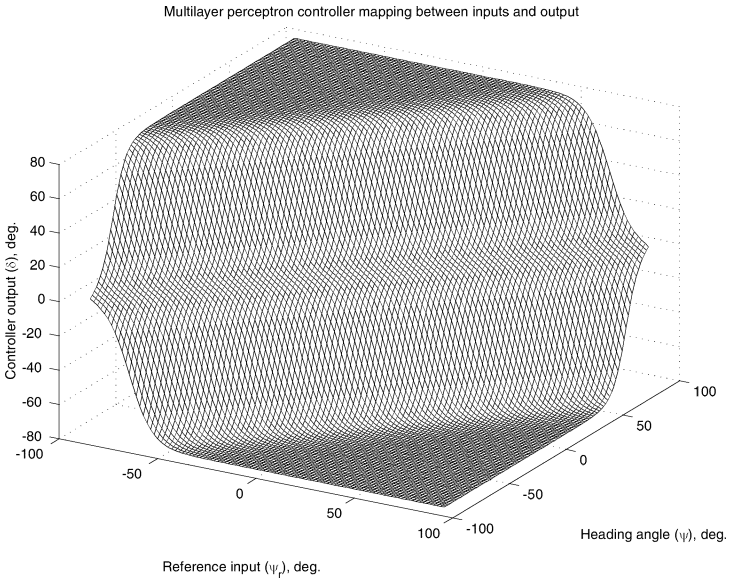

Figure 4.12: Control surface implemented by the multilayer perceptron for tanker ship steering.

The neural network stimulus-response mapping is generally nonlinear so it implements a nonlinear controller. To understand how this nonlinear controller might affect closed-loop behavior, it is important to have insights into the shape of the nonlinearity.

Figure 4.12 summarizes the input-output behavior of the multilayer perceptron. Consider some examples of how it behaves. Notice that if $e = \psi_r - \psi = 0$, the ship is heading in the correct direction, and the mapping in Figure 4.12 shows that the perceptron chooses $\delta = 0$ (i.e., it does not make any course corrections); this is due to the symmetry in our parameter choices for the top and bottom paths. If, on the other hand,

$$\psi_r = 50$$

degrees and

$$\psi = -50$$

degrees, then δ is at its maximum *negative* deflection so that it is trying to increase the heading angle ψ to get it pointed in the direction specified by ψ_r. Other combinations of values for ψ_r and ψ can be viewed in an analogous manner.

4.3.4 Behavior of the Ship Controlled by the Multilayer Perceptron

To evaluate how a neural network can regulate the ship heading, we will use simulation studies for a variety of operating conditions for the ship.

Closed-Loop Response, Nominal Conditions

If we use "nominal conditions," where we have "ballast" conditions, no wind, no sensor noise, and a speed of 5 meters/sec., we get a closed-loop response shown in Figure 4.13. For this, we use the multilayer perceptron in Figure 4.10 in the control system in Figure 4.9. Notice that the reference input ψ_r is set to zero for 100 sec. and then 45 degrees until $t = 2000$ sec. when it returns to a zero value. The actual ship heading responds quickly, but there is some overshoot past the desired value, some oscillations, and then the heading settles to the desired value. Note that the result of using a digital controller that only updates the control input every 10 sec. manifests itself as the "staircase" signal in the bottom plot of Figure 4.13. The middle plot also has a staircase form since we only stored and plotted one of every 10 values. The top plot appears smooth since we plot values each second.

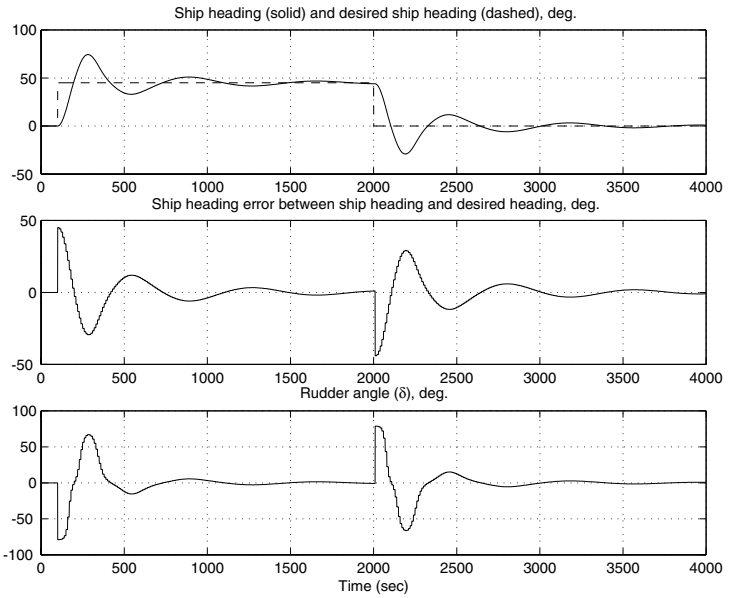

Figure 4.13: Closed-loop response resulting from using the multilayer perceptron for tanker ship steering.

This response may not be considered to be very good; however, for a first design it is reasonable. How do you improve the response? You tune the shape of the nonlinearity pictured in Figure 4.12 by tuning the weights and biases of the network, and possibly its structure (e.g., the number of layers, neurons, and types of activation functions). Another option would be to use more inputs to the controller, but we will consider this option when we study the use of a radial basis function neural network for this same application in Section 4.5.

For now, we will assume that this is a reasonably good design, at least for illustration purposes, and test its performance for other conditions.

Effects of Wind on Heading Regulation

Simulation-based evaluations of control systems should consider effects of a variety of adverse influences.

Next, consider the effects of a wind disturbance on the ship. Suppose that the wind is gusting. It hits the side of the ship and moves the ship a bit, which then pushes the rudder against the water which induces a torque to move the rudder. To model this, we add a disturbance onto the rudder angle input by adding

$$0.5\left(\frac{\pi}{180}\right)\sin\left(2\pi(0.001)t\right)$$

to what the multilayer perceptron controller commands as an input to the tanker ship (this is an additive sinusoid disturbance with an amplitude of 0.5 degrees and a period of 1000 sec.). In this case, we get the response in Figure 4.14. We see that the wind affects our ability to achieve very good regulation of the ship heading. In particular, it adversely affects the "steady-state behavior" of the control system (i.e., when the value of ψ_r is held constant for a long period of time, such as the times leading up to $t = 2000$ sec.) since the heading ψ does not properly converge to the desired heading ψ_r.

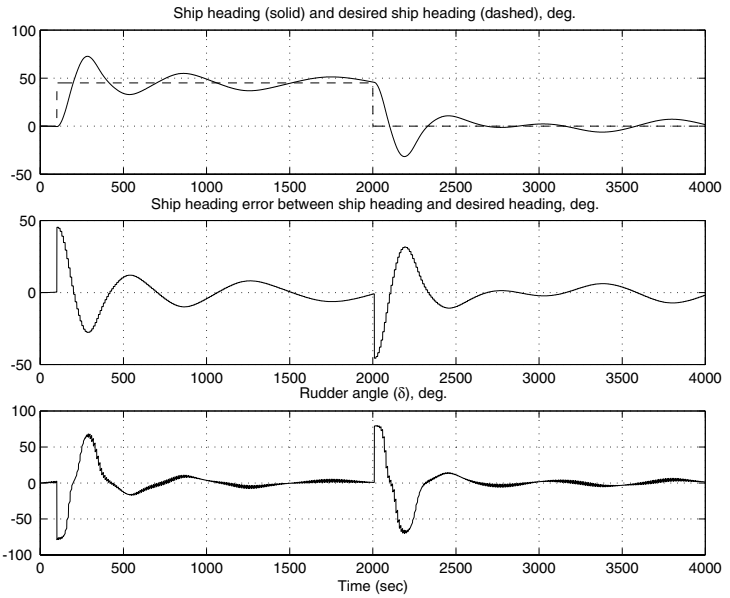

Figure 4.14: Closed-loop response resulting from using the multilayer perceptron for tanker ship steering, with wind.

Effects of Speed Changes on Heading Regulation

Next, consider the effect of a speed change on our ability to steer the ship. Generally, if you speed up the ship it is easier to steer, while if you slow it down, it becomes more difficult to steer because the rudder becomes less effective. If

we use a speed of $u = 3$ meters/sec. (i.e., a decreased speed compared to the previous simulations), then we get the response in Figure 4.15. We see that the speed decrease causes a general slowing of the response since the rudder is not as effective in influencing the ship heading.

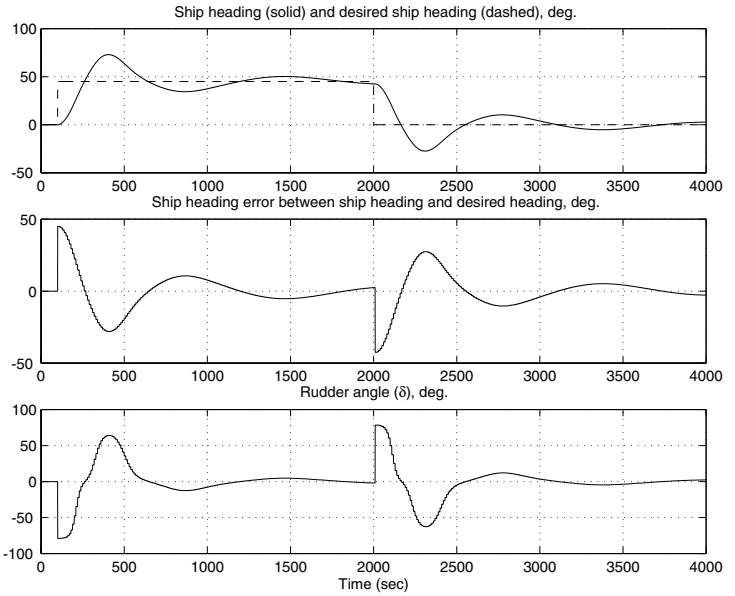

Figure 4.15: Closed-loop response resulting from using the multilayer perceptron for tanker ship steering with speed of 3 meters/sec.

Effects of Sensor Noise and Weight Changes on Heading Regulation

If you use an additive sensor noise uniformly distributed on $[-0.01, 0.01]$, there is little effect on the response so we do not show the plot. Of course, if you use a sensor with worse performance characteristics, then you will expect tracking errors to arise in an analogous manner to results for the wind.

Careful evaluations may uncover conditions for which the control system performs poorly.

Note that on different journeys, ships will weigh different amounts and the amount a ship weighs affects your ability to steer it. For the simulations up till now we have studied the case for "ballast" conditions. Next, we will consider the case of how the ship steers when it is under "full" conditions. In this case, when we use the multilayer perceptron that we tuned for ballast conditions on the full ship, we get the response in Figure 4.16. This shows how the multilayer perceptron controller, which was tuned for ballast conditions, performs quite poorly for full conditions. Why does it fail? It responds with too large of inputs for errors in the heading. In the beginning of the simulation when ψ_r first switches to 45 degrees at $t = 100$ sec. there is suddenly a positive error of $e = 45$ degrees, and the multilayer perceptron quickly reacts by putting in a maximum

negative value for the rudder to try to get it moving in the right direction. After a time it succeeds, but in doing this it has the ship heading moving too fast and it overshoots in the opposite direction. The multilayer perceptron then responds by putting a maximum positive value into the plant, which after an even longer period of time than in the case where the ψ value swung too far positive, it manages to move the ship heading in the opposite direction. This process repeats with the controller inducing a growing heading oscillation that results in the heading growing excessively large (which we intuitively think of as going "unstable," but of course strictly speaking, a simulation cannot generally prove instability since simulations run for a finite length of time while stability is an asymptotic property).

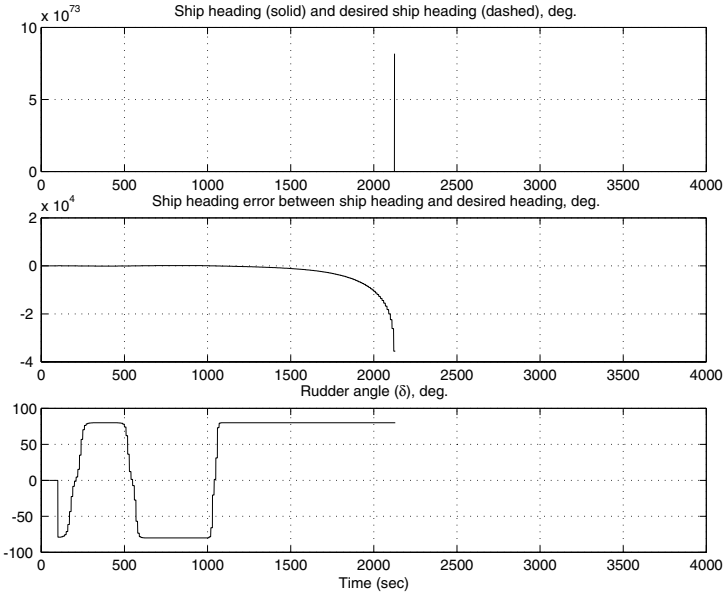

Figure 4.16: Closed-loop response resulting from using the multilayer perceptron for tanker ship steering, full rather than ballast conditions.

Clearly, the multilayer perceptron is not equipped for this condition. From a neurophysiological view, we could say that evolution has not encountered this situation frequently enough to result in a good design for the stimulus-response characteristics of the perceptron. From a control engineering perspective, we see that we need to reshape the nonlinearity $F_{mlp}(\psi_r, \psi)$ in Figure 4.12 so that the closed-loop response is adequate for all the possible conditions. For now, we will not consider performing such a design iteration. Instead, we will show how to use a different type of neural controller to regulate the ship heading using a different strategy.

4.4 Radial Basis Function Neural Networks

A locally tuned overlapping receptive field is found in parts of the cerebral cortex, in the visual cortex, and in other parts of the brain. The radial basis function neural network model is based on these biological systems (but once again, the model is not necessarily accurate, just inspired by its biological counterpart).

A radial basis function neural network is shown in Figure 4.17. There, the inputs are x_i, $i = 1, 2, \ldots, n$, and the output is $y = F_{rbf}(x)$ where F_{rbf} represents the processing by the entire radial basis function neural network. Let $x = [x_1, x_2, \ldots, x_n]^\top$. The input to the i^{th} receptive field unit (sometimes called a radial basis function) is x, and its output is denoted with $R_i(x)$. The receptive field unit has what is called a "strength" which we denote by b_i. Assume that there are n_R receptive field units. Hence, from Figure 4.17,

$$y = F_{rbf}(x, \theta) = \sum_{i=1}^{n_R} b_i R_i(x) \tag{4.10}$$

is the output of the radial basis function neural network, and θ holds the b_i parameters and possibly the parameters of the receptive field units.

Stimulus-response characteristics of the radial basis function neural network are tuned by changing the b_i, R_i parameters, or the structure (e.g., R_i definitions and how they are combined).

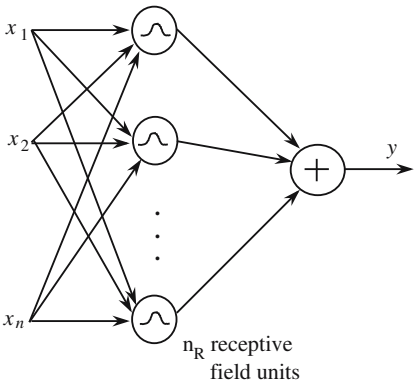

Figure 4.17: Radial basis function neural network model.

There are several possible choices for the "receptive field units" $R_i(x)$:

1. We could choose
$$R_i(x) = \exp\left(-\frac{|x - c^i|^2}{(\sigma^i)^2}\right) \tag{4.11}$$
where $c^i = [c_1^i, c_2^i, \ldots, c_n^i]^\top$, σ^i is a scalar, and if z is a vector then $|z| = \sqrt{z^\top z}$. For the case where $n = 1$, $c^1 = [c_1^1] = [2]$, and $\sigma^1 = 0.1$, $R_1(x)$ is shown in Figure 4.18(a). As x moves away from c_1^1, $R_1(x)$ decreases with the rate of decrease dictated by the size of σ^1 (a smaller value of σ^1 results in a steeper slope on the function, and so its value will decrease quicker as x moves away from c_1^1).

2. We could choose
$$R_i(x) = \frac{1}{1 + \exp\left(-\frac{|x-c^i|^2}{(\sigma^i)^2}\right)}$$
where c^i and σ^i are defined in choice 1. For the case where $n = 1$, $c^1 = [c_1^1] = [2]$, and $\sigma^1 = 0.1$, $R_1(x)$ is shown in Figure 4.18(b). Here, we see that the receptive field unit values are small where the values for choice 1 above are large, and vice versa.

3. In each of the above cases you can choose to make the σ^i also depend on the input dimension (which makes sense if the input dimensions are scaled differently). In this case for 1 above, for example, we would have $\sigma^i = [\sigma_1^i, \sigma_2^i, \ldots, \sigma_n^i]^\top$ and
$$R_i(x) = \exp\left(-\sum_{j=1}^{n} \frac{(x_j - c_j^i)^2}{(\sigma_j^i)^2}\right)$$
where σ_j^i is the spread for the j^{th} input for the i^{th} receptive field unit. This is the approach that we will use in the example in the next section.

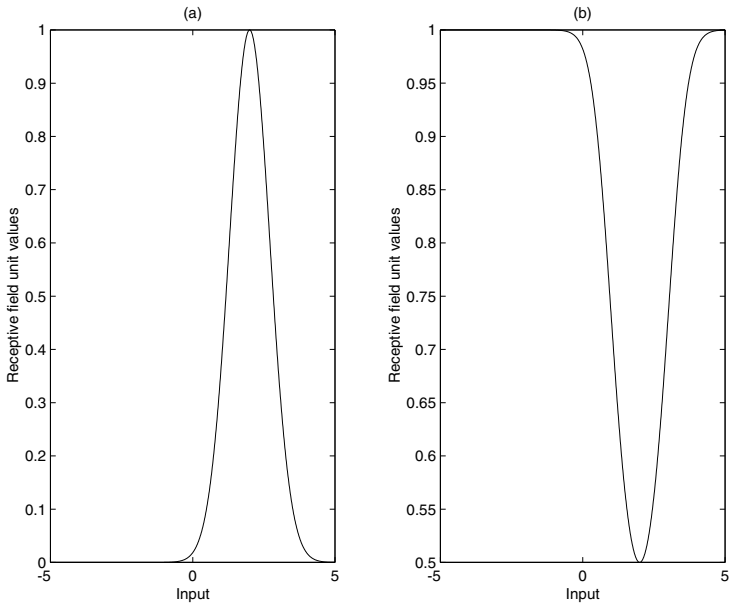

Figure 4.18: Example receptive field units.

There are also alternatives to how to compute the output of the radial basis function neural network. For instance, rather than computing the simple sum

as in Equation (4.10), you could compute a weighted average

$$y = F_{rbf}(x, \theta) = \frac{\sum_{i=1}^{n_R} b_i R_i(x)}{\sum_{i=1}^{n_R} R_i(x)} \tag{4.12}$$

It is also possible to define multilayer radial basis function neural networks.

Finally, note that our radial basis function neural network model is developed in an analogous way to what our multilayer perceptron was, relative to biological neurons. It is a "firing rate model" that has had the receptive field unit function shapes experimentally validated by finding the "tuning curve" for an individual neuron (e.g., in the visual cortex). See the "For Further Study" section at the end of this part for more details.

4.5 Design Example: Radial Basis Function Neural Network for Ship Steering

This section parallels Section 4.3, but we will design a radial basis function neural network for ship heading regulation.

4.5.1 A Radial Basis Function Neural Network for Ship Steering

We will design the radial basis function neural network, study its stimulus response characteristics, and then show via simulations how it regulates the tanker ship heading.

Controller Input Choice and Control System Structure

Note that for the multilayer perceptron, we used ψ_r and ψ as inputs to the controller and then in the first layer, we formed the error e that served as an input to the second layer. Here, taking a more standard control engineering approach, we will use the error e as one input to the radial basis function neural network, and we will also use the derivative of that error. Hence, our inputs to the radial basis function neural network will be

$$e = \psi_r - \psi$$

and

$$\dot{e} = \dot{\psi}_r - \dot{\psi}$$

We will, however, use a backward difference approximation to the derivative which we will denote by $c(kT)$,

$$\dot{e} \approx \frac{e(kT) - e(kT - T)}{T} = c(kT)$$

where $T = 10$ sec. and k is an index for the time step (this is an Euler approximation of the derivative and T is the sampling period of the digital controller

on which we will implement the controller). As is standard in discrete-time systems, for convenience we will often use "k" rather than "kT" as the argument for the signals. With this, we can denote the radial basis function neural network for the ship by

$$\delta(k) = F_{rbf}(e(k), c(k))$$

(we omit parameter vector argument for convenience) and use it in the control system shown in Figure 4.19.

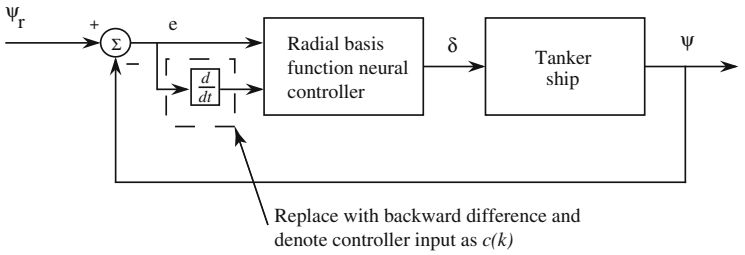

Figure 4.19: Radial basis function neural network used as a controller for ship heading.

Design of a Radial Basis Function Neural Network for Steering

Next, we construct a radial basis function neural network of Equation (4.10) with $n = 2$ inputs, and $n_R = 121$ so we will have to pick 121 strengths b_i, $i = 1, 2, \ldots, 121$. For the $R_i(e(k), c(k))$ we use Equation (4.11) and create a uniform grid for the c^i centers, $i = 1, 2, \ldots, 121$. To pick the grid points, assume that $e(k)$ lies in the range

$$e(k) \in [-\frac{\pi}{2}, \frac{\pi}{2}]$$

(which will hold if we do good regulation and do not get fast changes in ψ_r). Via simulations of the ship, the angular rate of movement is often such that

$$c(k) \in [-0.01, 0.01]$$

so we will make that assumption to guide our design choices. For convenience, we simply create a uniform grid with its four outer corners at $(-\frac{\pi}{2}, -0.01)$, $(-\frac{\pi}{2}, 0.01)$, $(\frac{\pi}{2}, 0.01)$, and $(\frac{\pi}{2}, -0.01)$ with $n_R = 121$ centers uniformly placed at the grid points (i.e., with 11 points along each input dimension). We show the centers of the receptive field units in Figure 4.20.

The parameters of receptive field units are often chosen via "gridding" the input space; this ensures that the controller will have a response for each input.

For the receptive field units we use spreads σ_j^i (i.e., so that the size of the spread depends on which input dimension is used) with

$$\sigma_1^i = 0.7\frac{\pi}{\sqrt{n_R}}$$

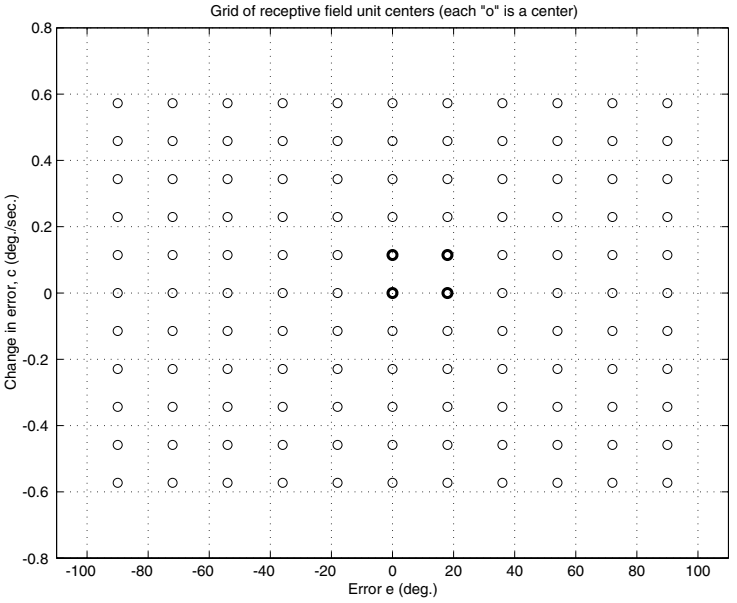

Figure 4.20: Receptive field unit centers.

and

$$\sigma_2^i = 0.7\frac{0.02}{\sqrt{n_R}}$$

for $i = 1, 2, \ldots, 121$. For σ_1^i, the $\frac{\pi}{11}$ factor makes the spread size depend on the number of grid points along the e input dimension (similarly for σ_2^i), and the 0.7 factor was chosen to get a smooth interpolation between adjacent receptive field units (see more discussion on this point below). With these choices, as an example, consider the shape of receptive field unit $R_{73}(e, c)$ shown in Figure 4.21 (note that the receptive field unit index is found by starting in the lower left-hand corner of Figure 4.20 with 1, and counting up for the point directly above it, then when you reach the top of the first column, you go to the bottom of the next column). Notice that it simply has the shape of a Gaussian function. The center of this particular receptive field unit is the upper right-hand darkly shaded circle in Figure 4.20. (When comparing Figures 4.20 and 4.21, be careful to mentally rotate Figure 4.20 so that the plane appropriately aligns with the $R_{73}(0, 0) = 0$ plane in Figure 4.21.)

We can view construction of a radial basis function neural network as building a stimulus-response characteristic from tunable "spatially local" functions (e.g., Gaussian functions).

Next, we will consider how the input-output mapping of the radial basis function neural network is shaped by the choice of the scaling parameters b_i. For instance, note that b_{73} would simply scale the height of the receptive field unit in Figure 4.21. Consider the scaling and summation of the receptive field units with centers at the four darkly shaded circles in Figure 4.20 (the indices for these are 61, 62, 72, and 73). In particular, we compute

$$2R_{61}(e, c) + R_{62}(e, c) + 2R_{72}(e, c) + R_{73}(e, c)$$

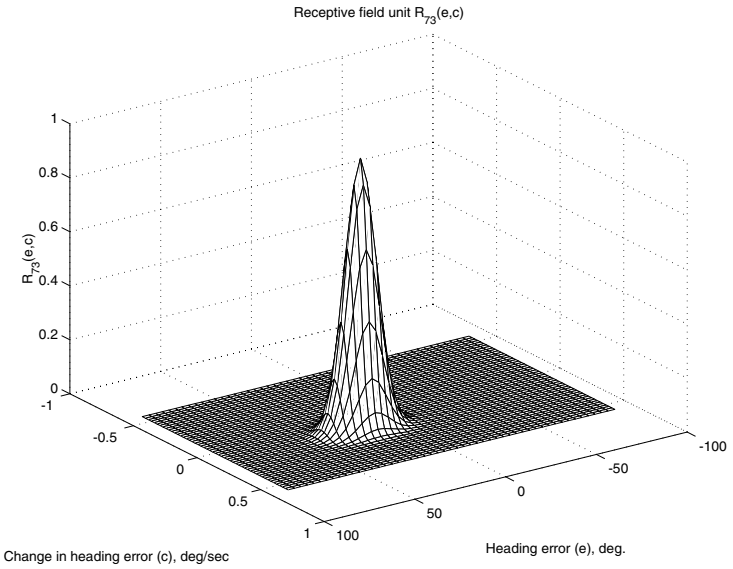

Figure 4.21: Mapping implemented by receptive field unit $R_{73}(e, c)$.

and plot it vs. e and c in Figure 4.22. Note that we scaled two of the receptive field units by 2 and in this way, we obtain a small region (near the center of the four darkly shaded circles in the (e, c) plane) that has a slope fixed by the parameters that we have chosen. In essence, we have designed a neural controller for this small region.

To design a radial basis function neural network for the ship steering problem, we simply need to choose the b_i, $i = 1, 2, \ldots, 121$, parameters to shape the mapping in the appropriate way. Suppose that we view the parameters as being loaded in a matrix

$$\begin{bmatrix} b_1 & b_{12} & b_{23} & b_{34} & b_{45} & b_{56} & b_{67} & b_{78} & b_{89} & b_{100} & b_{111} \\ b_2 & & & & & \cdots & & & & & b_{112} \\ \vdots & & & & & \cdots & & & & & \vdots \\ b_{11} & b_{22} & b_{33} & b_{44} & b_{55} & b_{66} & b_{77} & b_{88} & b_{99} & b_{110} & b_{121} \end{bmatrix}$$

and then choose this matrix to be

```
Columns 1 through 7

  1.3963    1.3963    1.3963    1.3963    1.3963    1.3963    1.3963
  1.3963    1.3963    1.3963    1.3963    1.3963    1.3963    1.0472
  1.3963    1.3963    1.3963    1.3963    1.3963    1.0472    0.6981
  1.3963    1.3963    1.3963    1.3963    1.0472    0.6981    0.3491
  1.3963    1.3963    1.3963    1.0472    0.6981    0.3491         0
  1.3963    1.3963    1.0472    0.6981    0.3491         0   -0.3491
```

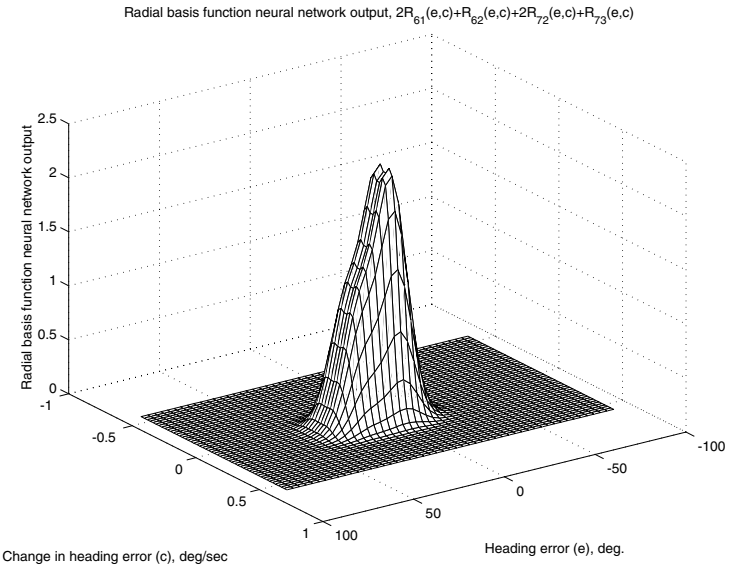

Figure 4.22: Scaling and addition of several receptive field units (i.e., $2R_{61}(e,c)+R_{62}(e,c)+2R_{72}(e,c)+R_{73}(e,c)$).

```
   1.3963    1.0472    0.6981    0.3491         0   -0.3491   -0.6981
   1.0472    0.6981    0.3491         0   -0.3491   -0.6981   -1.0472
   0.6981    0.3491         0   -0.3491   -0.6981   -1.0472   -1.3963
   0.3491         0   -0.3491   -0.6981   -1.0472   -1.3963   -1.3963
        0   -0.3491   -0.6981   -1.0472   -1.3963   -1.3963   -1.3963

  Columns 8 through 11

   1.0472    0.6981    0.3491         0
   0.6981    0.3491         0   -0.3491
   0.3491         0   -0.3491   -0.6981
        0   -0.3491   -0.6981   -1.0472
  -0.3491   -0.6981   -1.0472   -1.3963
  -0.6981   -1.0472   -1.3963   -1.3963
  -1.0472   -1.3963   -1.3963   -1.3963
  -1.3963   -1.3963   -1.3963   -1.3963
  -1.3963   -1.3963   -1.3963   -1.3963
  -1.3963   -1.3963   -1.3963   -1.3963
  -1.3963   -1.3963   -1.3963   -1.3963
```

Notice the pattern of elements in the matrix. For instance, for R_{61}, the receptive field unit in the center of the grid, we have a strength $b_{61} = 0$. Why? Because at this point $e = c = 0$ so the ship is on the proper heading and it is not deviating from that heading; hence, we do not make any corrections to the rudder angle. It is a useful exercise for you to consider another element in

the above matrix and convince yourself that it is a good choice via relating its choice to what the controller should do for a particular (e, c) combination.

4.5.2 Stimulus-Response Characteristics

The stimulus-response characteristics of the radial basis function neural network $F_{rbf}(e, c)$ that we just designed are shown in Figure 4.23 in the form of a control surface, similar to how we illustrated the mapping for the multilayer perceptron (note that here the inputs are different).

Different inputs and neural networks lead to different stimulus-response characteristics for the controller.

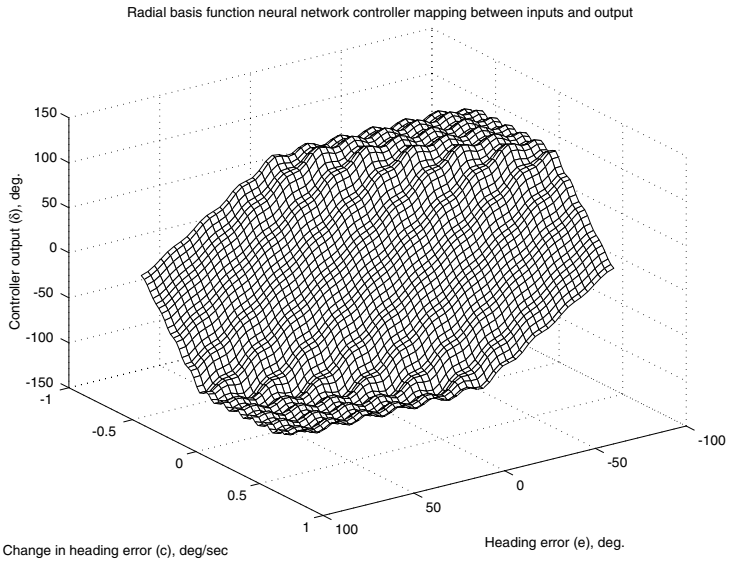

Figure 4.23: Stimulus-response characteristics of the radial basis function neural network for tanker ship heading regulation.

Note that the plot nicely summarizes the "decisions" that the neural network will make. Notice that if $e = c = 0$, the ship is heading in the proper direction and it is not deviating from that direction; hence, the controller sets $\delta = 0$. If, however, the ship heading error e is near 90 deg., with positive values of ψ and ψ_r, we know that the heading ψ is pointed about 90 deg. counterclockwise of the desired heading ψ_r. If along with this condition for e, we have that c is positive and near a value of 0.5 deg./sec., then the heading is moving to become even worse than it currently is. In this situation, the neural network will choose the largest possible *negative* rudder angle so that the heading will move clockwise towards the desired heading, counteracting the effects of having a rate of rotation in the wrong direction. For practice, it would be useful for you to consider other (e, c) values and explain why the decisions made by the neural controller are appropriate.

4.5.3 Behavior of the Ship Controlled by the Radial Basis Function Neural Network

To study how a radial basis function neural network can operate to regulate the ship heading, we will use simulation studies for a variety of operating conditions, the same ones as used in Section 4.3.

Closed-Loop Response, Nominal Conditions

If we use "nominal conditions," where we have "ballast" conditions, no wind, no sensor noise, and a speed of 5 meters/sec., we get a closed-loop response shown in Figures 4.24 and 4.25 when we use the radial basis function neural network developed in the last section in the control system in Figure 4.19. The ship heading ψ responds quickly, and while there is some overshoot past the desired value, the response settles to the proper value relatively quickly.

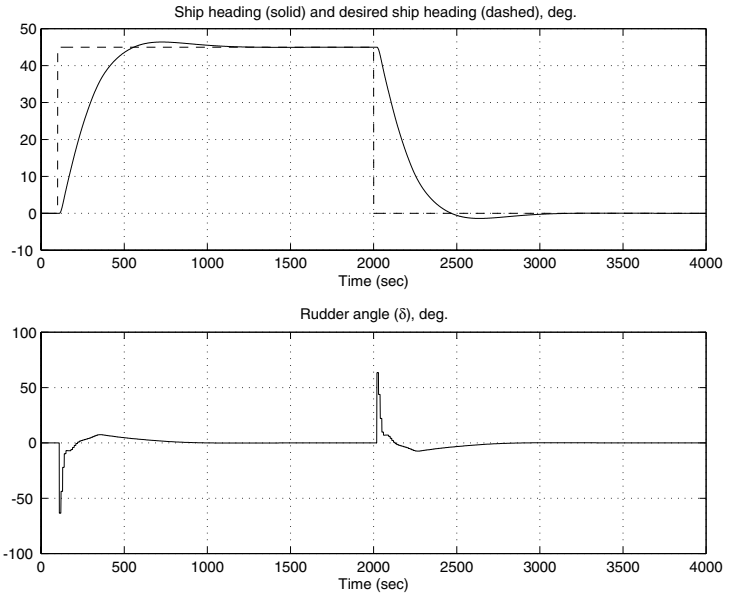

Figure 4.24: Closed-loop response resulting from using the radial basis function neural network for tanker ship steering.

While the response is generally superior to what we found for the multilayer perceptron, it is *not* appropriate to compare the two approaches. Why? Different inputs are used for the neural networks, there are far fewer parameters in the multilayer perceptron (how many were used in each case?), and we may have simply gotten lucky in our tuning for the radial basis function neural network.

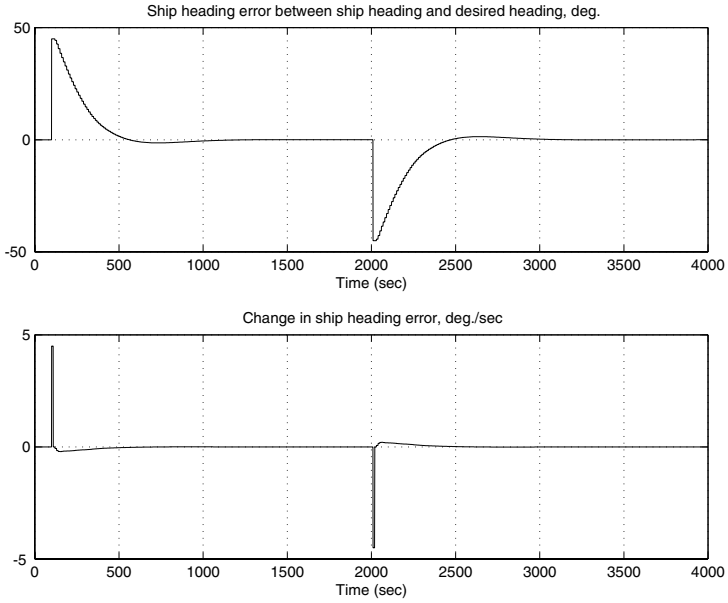

Figure 4.25: Closed-loop response resulting from using the radial basis function neural network for tanker ship steering.

Effects of Wind, Speed Changes, Sensor Noise, and Weight Changes on Heading Regulation

Next, consider the effects of a wind disturbance on the ship. In this case, we get the response in Figure 4.26. We see that the wind affects our ability to achieve very good steady-state regulation of the ship heading.

Next, consider the effect of a speed change on our ability to steer the ship. If we use a speed of $u = 3$ meters/sec. (i.e., a decreased speed compared to the previous simulations), then we get the response in Figure 4.27. We see that compared to the nominal conditions, the speed decrease causes more overshoot of the response since the rudder is not as effective in influencing the ship heading.

Using different inputs and a different neural network stimulus-response characteristic, we generally obtain different closed-loop responses.

As before, the sensor noise has little effect on the response. When there is a weight change and the ship is now full, we get the response in Figure 4.28. Notice that we get more overshoot than we did for nominal conditions; this is because a lighter ship is easier to steer so that the actions taken are too extreme and this results in the overshoot (you can think of the rudder as being more effective at steering for a light ship; hence, it generally needs smaller rudder inputs for a full ship). While the multilayer perceptron performed quite poorly for this condition, the radial basis function neural network performs reasonably well; however, just like for the nominal conditions above, it would be inappropriate to draw many conclusions from a comparison without more study. It would be

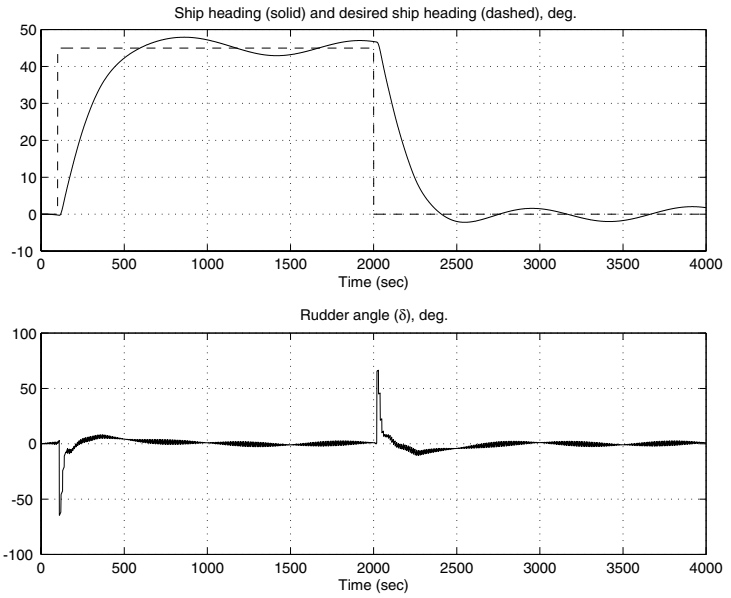

Figure 4.26: Closed-loop response resulting from using the radial basis function neural network for tanker ship steering, with wind.

especially inappropriate to try to conclude that the radial basis function neural network is generally superior to the multilayer perceptron.

4.6 Stability Analysis

Lyapunov stability analysis is an approach to verifying the correct operation of a control system.

For some applications, the designer is first concerned about investigating the stability properties of a control system, since it is often the case that if the system is unstable, there is no chance that any other performance specifications will hold. For example, if the control system for ship steering is unstable, you would be more concerned with the possibility of unsafe operation than with how well it regulates the heading to the desired angle. Fortunately, there has been significant attention given to the mathematical analysis of stability of nonlinear control systems, and certain results from that theory apply here. Here, we overview Lyapunov's direct method. For more complete introductions to stability analysis, see the "For Further Study" section at the end of this part.

4.6.1 Differential Equations and Equilibria

Suppose that a dynamic system is represented with

$$\dot{x}(t) = f(x(t)) \tag{4.13}$$

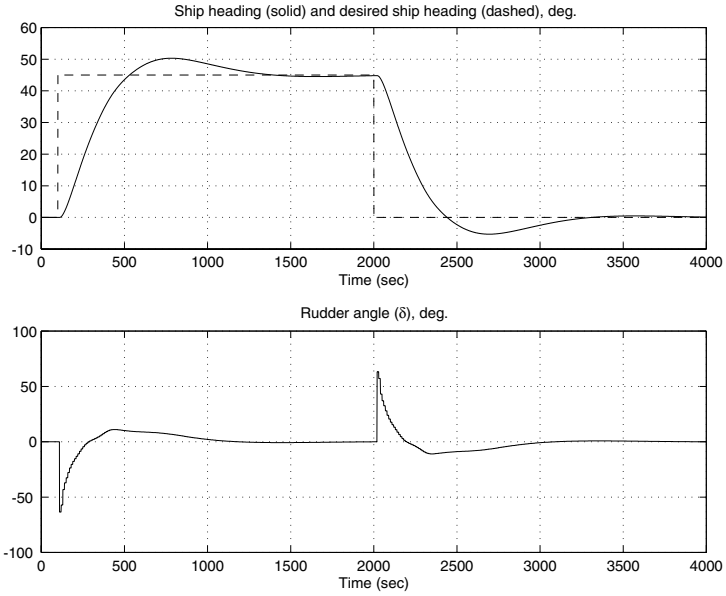

Figure 4.27: Closed-loop response resulting from using the radial basis function neural network for tanker ship steering, speed of 3 meters/sec.

where $x \in \Re^n$ is an n vector and $f : D \to \Re^n$ with $D = \Re^n$ or $D = B(h)$ for some $h > 0$ (h here is not to be confused with the integration step size used in the Runge-Kutta method) where

$$B(h) = \{x \in \Re^n : |x| < h\}$$

is a ball centered at the origin with a radius of h and $|\cdot|$ is a norm on $\Re^n$ (e.g., $|x| = \sqrt{(x^\top x)}$). If $D = \Re^n$, then we say that the dynamics of the system are defined globally, while if $D = B(h)$, they are only defined locally. Assume that for every x_0, the initial value problem

$$\dot{x}(t) = f(x(0)), \quad x(0) = x_0 \tag{4.14}$$

possesses a unique solution $\bar{\phi}(t, x_0)$ that depends continuously on x_0 ($\bar{\phi}(t, x_0)$ is a "solution" of Equation (4.13) if $\dot{\bar{\phi}}(t, x_0) = f(\bar{\phi}(t, x_0))$ where $\bar{\phi}(0, x_0) = x_0$). A point $x_e \in \Re^n$ is called an "equilibrium point" of Equation (4.13) if $f(x_e) = 0$ for all $t \geq 0$. An equilibrium point x_e is an "isolated equilibrium point" if there is an $h' > 0$ such that the ball around x_e,

$$B(x_e, h') = \{x \in \Re^n : |x - x_e| < h'\}$$

contains no other equilibrium points besides x_e. As is standard, we will assume that the equilibrium of interest is an isolated equilibrium located at the origin of $\Re^n$. This assumption results in no loss of generality since if $x_e \neq 0$ is an

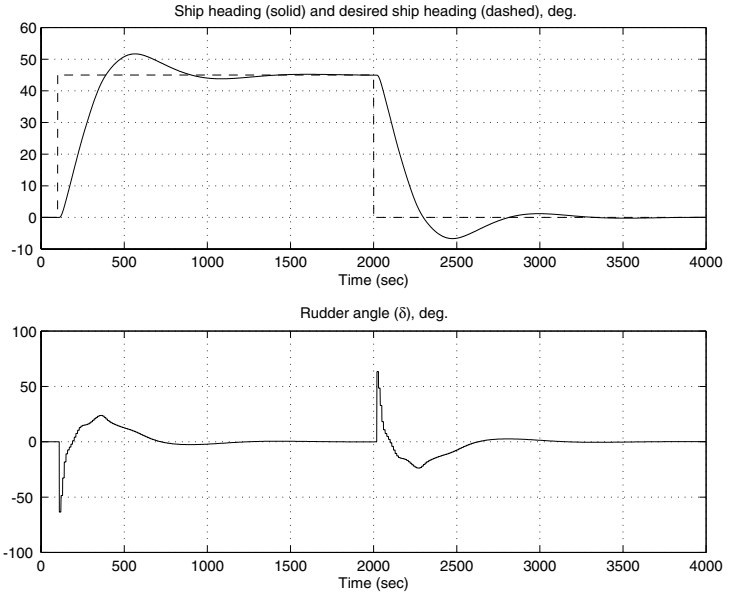

Figure 4.28: Closed-loop response resulting from using the radial basis function neural network for tanker ship steering, full rather than ballast conditions.

equilibrium of Equation (4.13) and we let $\bar{x}(t) = x(t) - x_e$, then $\bar{x} = 0$ is an equilibrium of the transformed system

$$\dot{\bar{x}}(t) = \bar{f}(\bar{x}(t)) = f(\bar{x}(t) + x_e)$$

To illustrate how to transform the equilibrium, we use a simple model of the pendulum shown in Figure 4.29 that is given by

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -\tfrac{g}{\ell}\sin(x_1) - \tfrac{k}{m}x_2 + \tfrac{1}{m\ell^2}T \end{aligned} \tag{4.15}$$

where $g = 9.81$, $\ell = 1.0$, $m = 1.0$, $k = 0.5$, x_1 is the angle (in radians) shown in Figure 4.29, x_2 is the angular velocity (in radians per second), and T is the control input.

If we assume that $T = 0$, then there are two distinct isolated equilibrium points, one in the downward position $[0, 0]^\top$ and one in the inverted position $[\pi, 0]^\top$. Suppose we are interested in the control of the pendulum about the inverted position; hence, we need to translate the equilibrium by letting $\bar{x} = x - [\pi, 0]^\top$. From this we obtain

$$\begin{aligned} \dot{\bar{x}}_1 &= \bar{x}_2 = \bar{f}_1(\bar{x}) \\ \dot{\bar{x}}_2 &= \tfrac{g}{\ell}\sin(\bar{x}_1) - \tfrac{k}{m}\bar{x}_2 + \tfrac{1}{m\ell^2}T = \bar{f}_2(\bar{x}) \end{aligned} \tag{4.16}$$

where if $T = 0$, then $\bar{x} = 0$ corresponds to the equilibrium $[\pi, 0]^\top$ in the original system in Equation (4.15), so studying the stability of $\bar{x} = 0$ corresponds to

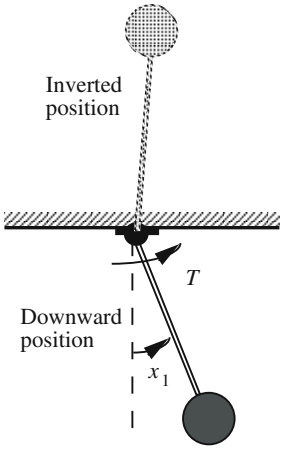

Figure 4.29: Pendulum.

studying the stability of the control system about the inverted position. Now, it is traditional to omit the cumbersome bar notation in Equation (4.16) and study the stability of $x = 0$ for the system

$$\begin{aligned} \dot{x}_1 &= x_2 = f_1(x) \\ \dot{x}_2 &= \tfrac{g}{\ell}\sin(x_1) - \tfrac{k}{m}x_2 + \tfrac{1}{m\ell^2}T = f_2(x) \end{aligned} \tag{4.17}$$

with the understanding that we are actually studying the stability of Equation (4.16).

4.6.2 Stability Definitions

The equilibrium $x_e = 0$ of Equation (4.13) is "stable" (in the sense of Lyapunov) if for every $\epsilon > 0$ there exists a $\delta(\epsilon) > 0$ such that $|\bar{\phi}(t, x_0)| < \epsilon$ for all $t \geq 0$ whenever $|x_0| < \delta(\epsilon)$ (i.e., it is stable if when it starts close to the equilibrium, it will stay close to it). The notation $\delta(\epsilon)$ means that δ depends on ϵ. A system that is not stable is called "unstable."

The equilibrium $x_e = 0$ of Equation (4.13) is said to be "asymptotically stable" if it is stable and there exists $\eta > 0$ such that $\lim_{t\to\infty} \bar{\phi}(t, x_0) = 0$ whenever $|x_0| < \eta$ (i.e., it is asymptotically stable, if when it starts close to the equilibrium, it will converge to it).

The set $X_d \subset \Re^n$ of all $x_0 \in \Re^n$ such that $\bar{\phi}(t, x_0) \to 0$ as $t \to \infty$ is called the "domain of attraction" of the equilibrium $x_e = 0$ of Equation (4.13). The equilibrium $x_e = 0$ is said to be "globally asymptotically stable" if $X_d = \Re^n$ (i.e., if no matter where the system starts, its state converges to the equilibrium asymptotically).

As an example, consider the scalar differential equation

$$\dot{x}(t) = -2x(t)$$

which is in the form of Equation (4.14). For this system, $D = \Re^1$ (i.e., the dynamics are defined on the entire real line, not just some region around zero). We have $x_e = 0$ as an equilibrium point of this system since $0 = -2x_e$. Notice that for any x_0, we have the solution

$$\bar{\phi}(t, x_0) = x_0 e^{-2t} \to 0$$

as $t \to \infty$ so that the equilibrium $x_e = 0$ is stable since, if you are given any $\epsilon > 0$, there exists a $\delta > 0$ such that if $|x_0| < \delta$, $|\bar{\phi}(t, x_0)| < \epsilon$. To see this, simply choose $\delta = \epsilon$ for any $\epsilon > 0$ that you choose. Also note that since for any $x_0 \in \Re^n$, $\bar{\phi}(t, x_0) \to 0$, the system is globally asymptotically stable. While determining if this system possesses certain stability properties is very simple since the system is so simple, for complex nonlinear systems it is not so easy. One reason why is that for complex nonlinear systems, it is difficult to even solve the ordinary differential equations (i.e., to find $\bar{\phi}(t, x_0)$ for all t and x_0). However, Lyapunov's direct method provides a technique that allows you to determine stability properties without solving the ordinary differential equations.

4.6.3 Lyapunov's Direct Method for Stability Analysis

The stability results for an equilibrium $x_e = 0$ of Equation (4.13) that we provide next depend on the existence of an appropriate "Lyapunov function"

$$V : D \to \Re$$

where $D = \Re^n$ for global results (e.g., global asymptotic stability) and $D = B(h)$ for some $h > 0$, for local results (e.g., stability in the sense of Lyapunov or asymptotic stability). If V is continuously differentiable with respect to its arguments, then the derivative of V with respect to t along the solutions of Equation (4.13) is

$$\dot{V}_{(4.13)}(x(t)) = \nabla V(x(t))^\top f(x(t))$$

where

$$\nabla V(x(t)) = \left[\frac{\partial V}{\partial x_1}, \frac{\partial V}{\partial x_2}, \ldots, \frac{\partial V}{\partial x_n}\right]^\top$$

is the gradient of V with respect to x. Using the subscript on $\dot{V}$ is sometimes cumbersome, so we will at times omit it with the understanding that the derivative of V is taken along the solutions of the differential equation.

Lyapunov's direct method is given by the following:

1. Let $x_e = 0$ be an equilibrium for Equation (4.13). Let $V : B(h) \to \Re$ be a continuously differentiable function on $B(h)$ such that $V(0) = 0$ and $V(x) > 0$ in $B(h) - \{0\}$, and $\dot{V}_{(4.13)}(x) \leq 0$ in $B(h)$. Then $x_e = 0$ is stable. If, in addition, $\dot{V}_{(4.13)}(x) < 0$ in $B(h) - \{0\}$, then $x_e = 0$ is asymptotically stable.

2. Let $x_e = 0$ be an equilibrium for Equation (4.13). Let $V : \Re^n \rightarrow \Re$ be a continuously differentiable function such that $V(0) = 0$ and $V(x) > 0$ for all $x \neq 0$, $|x| \rightarrow \infty$ implies that $V(x) \rightarrow \infty$, and $\dot{V}_{(4.13)}(x) < 0$ for all $x \neq 0$. Then $x_e = 0$ is globally asymptotically stable.

As an example, consider the scalar dynamical system

$$\dot{x} = -2x^3$$

that has an equilibrium $x_e = 0$. Choose

$$V(x) = \frac{1}{2}x^2$$

With this choice we have

$$\dot{V} = \frac{\partial V}{\partial x}\frac{dx}{dt} = x\dot{x} = -2x^4$$

so that clearly if $x \neq 0$, then $-2x^4 < 0$, so that by Lyapunov's direct method $x_e = 0$ is asymptotically stable. Notice that $x_e = 0$ is in fact globally asymptotically stable.

While Lyapunov's direct method has found wide application in conventional control, it is important to note that it is not always easy to find the "Lyapunov function" V that will have the above properties so that we can guarantee that the system is stable.

4.6.4 Stability of Discrete Time Systems

Consider the nonlinear discrete time system

$$x(k+1) = f(x(k)) \tag{4.18}$$

where k is the discrete time index, $x \in \Re^n$ is an n vector and $f : D \rightarrow \Re^n$ (with $D = \Re^n$ or $D = B(h)$ for some $h > 0$), and the equilibrium $x_e \in \Re^n$ is defined the same as in the continuous time case. Let $\bar{\phi}(k, x_0)$ denote a solution to the nonlinear discrete time system where $x_0 = x(0)$.

Stability in the sense of Lyapunov, (global) asymptotic stability, and regions of asymptotic stability are defined the same as in the continuous time case, except the time index "t" is replaced with the index "k."

Stability conditions for the discrete-time direct method of Lyapunov are slightly different from the continuous time case. We will only discuss asymptotic stability, as that property will be the one we are most interested in for our applications. According to Lyapunov's direct method, the equilibrium $x_e = 0$ of the system in Equation (4.18) is globally asymptotically stable if there exists a function $V(x)$ such that the following hold for all $x \in \Re^n$:

1. $V(x) \geq 0$ except at $x = 0$ where $V(x) = 0$,
2. $V(x) \rightarrow \infty$ if $|x| \rightarrow \infty$, and

3. $V(x(k+1)) - V(x(k)) < 0$.

If these conditions only hold locally, then we only obtain asymptotic stability. If they only hold on a region, that region is the region of asymptotic stability. Also, if x_e is an invariant set (i.e., where if we let $x_0 \in x_e$, $f(x_0) \in x_e$), then the same types of results hold (we will give an example of how to perform such analysis in the examples to follow).

As an example, consider

$$x(k+1) = ax(k)$$

where a is a fixed scalar and $x(k)$ is a scalar also. Notice that $x_e = 0$ is an isolated equilibrium. Suppose we want to find the conditions under which $x_e = 0$ is a globally asymptotically stable equilibrium. Choose $V = x^2$. Notice that the first two conditions above are satisfied for this choice. Next, notice that

$$V(x(k+1)) - V(x(k)) = x^2(k+1) - x^2(k) = a^2x^2 - x^2 = (a^2 - 1)x^2$$

Hence, if $a^2 - 1 < 0$, we have $V(x(k+1)) - V(x(k)) < 0$. In other words, if $a^2 < 1$, or if $a \in (-1, 1)$, then $x_e = 0$ is a globally asymptotically stable equilibrium.

4.6.5 Example: Stable Instinctual Neural Control

Suppose you are given the differential equation

$$\dot{x} = f(x) + gu$$

where $x(t)$ is a scalar, $g > 0$ is an unknown but fixed scalar (the following analysis works in a similar way if we know that $g < 0$), f is smooth (so solutions to the differential equation exist and are unique), and $f(0) = 0$. We will assume that while we do not know the exact form of $f(x)$, we do suppose that for some $\alpha > 0$,

$$|f(x)| < \alpha|x|$$

Lyapunov stability analysis is useful to verify the correct operation of a control system using an instinctual neural controller.

We emphasize, however, that there is *uncertainty* present in this control problem in the sense that we do not know the value of g and we do not know the specific form of the nonlinearity f, just that it satisfies the above inequality.

We seek to design a neural controller

$$u = F(x)$$

so that the equilibrium $x_e = 0$ is globally asymptotically stable. First, pick

$$V(x) = \frac{1}{2}x^2$$

so that

$$\dot{V} = x\dot{x}$$

and so

$$\dot{V} = xf(x) + gxu = xf(x) + gxF(x)$$

Notice that

$$\dot{V} \leq |x||f(x)| + gxF(x) \leq \alpha x^2 + gxF(x)$$

We want to design the neural controller $F(x)$ so that the second term in the above $\dot{V}$ equation is negative since then we will have $\dot{V} < 0$ for $x \neq 0$ and then $x_e = 0$ will be a globally asymptotically stable equilibrium. To do this, suppose that we design the controller so that $F(0) = 0$, $F(x)$ is smooth, and for some scalar $\beta > 0$,

$$\begin{aligned} F(x) &> -\beta x,\ x < 0 \\ F(x) &< -\beta x,\ x > 0 \end{aligned} \tag{4.19}$$

which simply constrains the nonlinear surface of the neural controller. Now, if $x > 0$, $F(x) < -\beta x$, so

$$\dot{V} \leq \alpha x^2 + gx(-\beta x) = (\alpha - g\beta)x^2$$

Also, if $x < 0$, $F(x) > -\beta x$, so once again

$$\dot{V} \leq \alpha x^2 - g\beta x^2 = (\alpha - g\beta)x^2$$

Hence, if we have $\alpha - g\beta < 0$ or $\beta > \alpha/g$, then $x_e = 0$ will be globally asymptotically stable.

So, intuitively, why does our neural controller stabilize this uncertain nonlinear plant? Basically, when $x > 0$, $F(x) < 0$ so the neural controller seeks to make the derivative $\dot{x}$ negative to get the state x to move toward $x_e = 0$. Similarly, if $x < 0$, $F(x) > 0$ so the neural controller seeks to make the derivative $\dot{x}$ positive to get the state to move toward $x_e = 0$. It should be clear that it is not necessary for $F(x)$ to be a neural controller to achieve the stabilization task; any controller that satisfies the conditions in Equation (4.19) (and the other constraints) will adequately perform the task.

All of this analysis is based on our ability to synthesize a neural controller so that Equation (4.19) is met. To do this, you would need to write out the mathematical form of $F(x)$ and prove that it satisfies Equation (4.19); perhaps in this simple case, you could use a somewhat heuristic graphical technique where you construct the neural controller and plot its surface to check Equation (4.19). Notice that for many neural controllers, the output saturates for some large magnitude values of x so that Equation (4.19) will often not be satisfied globally. In this case, the analysis is not global, but only for an interval of the x axis, so we can only conclude that x_e is asymptotically stable (i.e., a local property) or that there is some region of asymptotic stability.

4.7 Hierarchical Neural Networks

There are a variety of methods that can be employed to construct hierarchical neural networks. Here, we provide an example of how such hierarchies occur in

nature, then discuss how multilayer perceptrons and radial basis function neural networks can be organized in a hierarchical fashion.

It is natural to view some neural networks as hierarchical.

4.7.1 Example: Marine Mollusc

In the marine mollusc, *Pleurobranchaea*, behaviors are organized hierarchically as dictated by the cellular arrangement of their neurons [312]. The arrangement (see Figure 472 in [312]), shows that the "swimming escape response" inhibits the other behaviors. Also, egg laying inhibits feeding, which in turn takes precedence over mating. The actual neural "circuitry" has been traced in these molluscs and this research has shown that when activated, command systems of neurons that are responsible for feeding and egg laying inhibit the neural networks dedicated to mating and locomotion.

The behavior of the mollusc is directly dictated by the underlying hierarchical organization of its neural network. Evolution has shaped a hierarchical arrangement in the neural network so that the behaviors that are exhibited increase the reproductive success of the mollusc.

4.7.2 Hierarchical Neural Structures

Here, we simply provide some ideas on how to structure neural networks in a hierarchical fashion. First, you could use a multilayer perceptron to turn on and off different parts of another multilayer perceptron. For example, the higher layer could simply output zeros and ones and these could multiply activation function outputs so that the lower level perceptron is reconfigurable based on different conditions.

For radial basis functions you may have a two-level hierarchical network with the higher layer defined on a coarse grid and the lower layer on a fine grid. Then, when a region is activated in the higher level network, that could activate a radial basis function neural network that is defined on a fine grid. This provides a type of "nesting" and focusing, and at times can provide for savings in computational complexity since only those radial basis functions with fine grids that are activated need to be stored in memory and computed.

4.8 Exercises and Design Problems

Exercise 4.1 (Building Multilayer Perceptrons): In this problem you will focus on constructing, "by hand," multilayer perceptrons to match certain functions.

(a) Construct two different multilayer perceptrons that try to match (approximate) the input-output properties of $y = f(x) = 2x$ where x and y are scalars over the range $x \in [-10, 10]$. In the first case you may use a linear activation function, and any combination of other neurons. In the second case, use a linear activation function in the output layer and a single hidden layer of no more than five logistic

activation functions. Try to tune the parameters of the network *by hand* to make the mapping that is implemented by the neural network as close as possible to f over its entire domain. Do not use the neural network training methods that are introduced later in the book. Plot f vs. x and the mapping implemented by the neural network on the same plot in order to illustrate how close the network approximates the function.

(b) Repeat (a) but for $y = f(x) = 2x^2$. You may use any type of multilayer perceptron, with any number of neurons you would like.

(c) Repeat (a) but for $y = f(x) = 2\sin(x)$. You may use any type of multilayer perceptron, with any number of neurons you would like.

Exercise 4.2 (Building Radial Basis Function Neural Networks):

(a) Repeat Exercise 4.1 (a), but only construct one radial basis function neural network with no more than five receptive field units.

(b) Repeat Exercise 4.1 (b), but only construct one radial basis function neural network with no more than five receptive field units.

(c) Repeat Exercise 4.1 (c), but only construct one radial basis function neural network and you may use any number of receptive field units for it.

Exercise 4.3 (Lyapunov's Direct Method): Suppose that you are given the plant

$$\dot{x} = ax + bu$$

where $b > 0$ and $a < 0$ (so the system is stable) and x is a scalar. Suppose that you design an instinctual neural controller F that generates the input to the plant given the state of the plant (i.e., $u = F(x)$). Assume that you design the controller so that $F(0) = 0$ (so that $x = 0$ is an equilibrium) and so that $F(x)$ is continuous in x (so that a unique solution exists to the differential equation describing the closed-loop system).

(a) Use Lyapunov's direct method to show that if x and $F(x)$ always have opposite signs, then $x = 0$ is stable.

(b) What types of stability does $x = 0$ of the control system possess for part (a)? List all types of stability that it possesses.

(c) Design a (SISO) instinctual neural controller that satisfies the condition stated in (a) (and so that $F(0) = 0$ and $F(x)$ is continuous) and simulate the closed-loop system to help illustrate the stability of the neural control system. Choose the initial condition $x(0) = 1$, $a = -2$, and $b = 2$. Of course, the simulation does not *prove* that the closed-loop system is stable—it only shows that for one initial condition, the state appears to converge but cannot prove that it converges since the simulation is only for a finite amount of time.

Exercise 4.4 (Multilayer Perceptron for Tanker Ship Steering): Produce simulations to reproduce the results where we used a multilayer perceptron for tanker ship steering in the chapter (all the conditions). Add more comments to the code and produce a flowchart to demonstrate that you understand its operation.

Exercise 4.5 (Radial Basis Function Neural Network for Tanker Ship Steering): Produce simulations to reproduce the results where we used a radial basis function neural network for tanker ship steering in the chapter (all the conditions). Add more comments to the code and produce a flowchart to demonstrate that you understand its operation.

Design Problem 4.1 (Design of a Multilayer Perceptron for Tanker Ship Steering):

(a) Redesign the multilayer perceptron from Exercise (4.4) to improve performance of the closed-loop system for nominal conditions. Constrain the way that you perform the redesign to simply tuning of parameters, not changing the number of layers or neurons. Show plots to support your conclusions.

(b) Repeat (a) but design a multilayer perceptron that has two inputs, e and $\dot{e}$ (that you may approximate using an Euler approximation to the derivative), and one output δ. Hint: Build on the multilayer perceptron that was used in (a). Tune the multilayer perceptron so that it obtains "better" performance (you define precisely what this means for your study) than in (a). Plot the three-dimensional input-output map of the resulting tuned controller.

(c) Repeat (a) but you may use any type of multilayer perceptron (i.e., you choose the inputs, number of layers, and neurons). Try to achieve the best possible performance for all the different conditions considered in the chapter. You define what you mean by good performance, and you decide what an appropriate balance is in the quality of the results between the different conditions.

Design Problem 4.2 (Design of a Radial Basis Function Neural Network for Tanker Ship Steering):

(a) Redesign the radial basis function neural network from Exercise (4.5) to improve performance of the closed-loop system for nominal conditions. Constrain the way that you perform the redesign to simply tuning of parameters, not changing the number of receptive field units. Show plots to illustrate better performance. Plot the three-dimensional input-output map of the resulting tuned controller.

(b) Repeat (a), but you may use any radial basis function neural network (i.e., you choose the inputs and number of receptive field units).

(c) Repeat (a) with the modification in (b), but try to achieve the best possible performance for all the different conditions considered in the chapter. You define what you mean by good performance, and you decide what an appropriate balance is in the quality of the results between the different conditions.