Multi–Stage Neural Networks with Application to Motion Planning of a Mechanical Snake

Barış Fidan*, Erol Sezer†, and Mehmet Akar*

Abstract

An efficient approach to nonlinear control problems in which the plant is to be driven to a desired state using a neural network controller in a number of steps is by representing the whole process as a multi–stage neural network. In this paper, an explicit formulation of the back propagation algorithm is developed for such networks. Later, this approach is used to build up a path planner for a mechanical snake (a robot composed of a sequence of articulated links). This path planner, together with a tracking algorithm, is shown to get the mechanical snake out of a collision–free closed region.

I Introduction

A *multi–stage neural network* is a large–scale neural network which is composed of a number of cascaded smaller scale networks. Such a structure is useful when the task of the neural network to be designed can be decomposed into simpler subtasks. Then, simple neural networks can be designed for these subtasks and combined as a single network in order to perform the overall task. The multi–stage neural network structure is also useful for control problems where the controller has to drive the plant in a number of steps to achieve a desired task, for which only the desired output for the final step is available.

In [1, 2], Nguyen and Widrow developed a multi–stage neural network approach for controlling nonlinear dynamic systems, and applied it to the *Truck Backer–Upper Problem*. In this application, the aim is to design a controller to back a trailer truck to a loading dock by only backing (i.e., going forward is not allowed) starting from an arbitrary initial position. In [1], the plant is approximated by an emulator neural network NN_D, and the process is assumed to end in K steps, where K is a fixed design constant. The K–step process is represented as a K–stage neural network composed of K cascaded identical NN_C–NN_D pairs as shown in Fig. 1, where NN_C is the controller neural network. The weights of NN_C are adjusted by applying the back propagation algorithm to the K–stage neural network structure. In [1, 2], it is demonstrated via simulation results that the proposed approach is effective in training neural network controllers, however some of the error terms in the method are not well defined and there is no explicit formulation of the back propagation through K–stages. In this paper, we first formulate a slightly different approach to apply back propagation in multi–stage neural networks with identical stages. Then, we derive explicit update equations to use back propagation effectively in such networks. Finally, the approach is applied to motion planning of a mechanical snake, which is a robot with a structure and motion pattern similar to those of a snake.

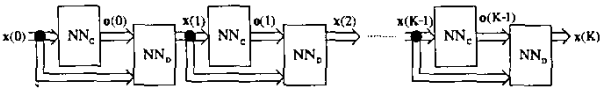

Figure 1: K–stage neural network representation of a K–step process.

A mechanical snake is useful in situations where it is necessary to crawl into places that are too dangerous or too narrow for people to enter, such as disaster areas and nuclear power plants with narrow vessels [3]. A typical task for a mechanical snake is to generate smooth minimal paths that can be tracked unobtrusively, to exit an enclosed area, and to track the generated paths efficiently. In this paper, this problem is studied for a collision–free environment.

The rest of the paper is organized as follows. In Section 2, multi–stage neural networks are introduced formally, and an explicit formulation of the back propagation algorithm through K–stages is developed. In Section 3, the task of motion planning for a mechanical snake is discussed, and the problem of exiting from an obstacle–free closed region is presented. Section 4 is devoted to the solution of this problem using the multi–stage neural network approach described in Section 2. Finally, some simulation results are presented in Section 5 and concluding remarks are given in Section 6.

II Multi–Stage Neural Networks with Identical Stages

Consider a nonlinear discrete–time plant

$$\mathbf{x}(k+1) = D(\mathbf{x}(k), \mathbf{o}(k)) \qquad (2.1)$$

where $\mathbf{x}(k) \in \Re^n$ is the state, and $\mathbf{o}(k) \in \Re^m$ is the control input. A common problem for this plant is

* *Department of Electrical Engineering–Systems, University of Southern California*, Los Angeles, CA 90089-2562.

† *Department of Electrical and Electronics Engineering, Bilkent University*, Ankara, 06533, Turkey.

0-7803-7896-2/03/$17.00 ©2003 IEEE

Proceedings of the American Control Conference
Denver, Colorado June 4-6, 2003

to choose the input sequence $\mathbf{o}$ which would drive the plant from an initial state $\mathbf{x}(0)$ to a desired state $\mathbf{d}$ in a certain number of steps. Letting the input $\mathbf{o}(k)$ to be a function of the state $\mathbf{x}(k)$ at each step k, i.e.,

$$\mathbf{o}(k) = C(\mathbf{x}(k)) \tag{2.2}$$

this problem is equivalent to choosing an appropriate function $C(\cdot)$. In this paper we focus on building a 2-layer neural network NN_C, as shown in Fig. 2, to implement the function $C(\cdot)$. For easier training of NN_C, the plant function $D(\cdot)$ has to be smooth (with a continuous first derivative). If it is not smooth, it can be approximated by a smooth function, which can be obtained either by direct derivation or using a neural network NN_D to emulate the plant. Some methods to emulate plants using multilayer neural networks are given in [1, 2, 4].

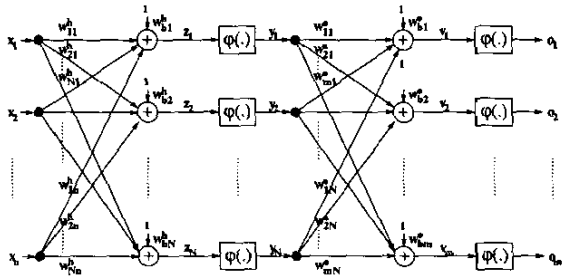

Figure 2: The controller neural network NN_C.

To train NN_C, we use a training set composed of sample initial states and desired final states. The number of steps to reach the desired state for each initial state is unknown and this number is in general not the same for all initial states. We can directly obtain only the final errors using the final states and their desired values. For each initial state, desired state pair $(\mathbf{x}(0), \mathbf{d})$, starting with $\mathbf{x}(0)$, the system is run until a stopping criterion holds. Consider the process for a specific pair in the training set. Assuming that the process lasts for K steps, it can be represented by the K-stage neural network structure shown in Fig. 1. The goal of the training process is to minimize the error

$$\epsilon = \frac{1}{2}|\mathbf{x}(K) - \mathbf{d}|^2 = \frac{1}{2}\sum_{i=1}^{n}(x_i(K) - d_i)^2 \tag{2.3}$$

where $\mathbf{x} = [x_1, \cdots, x_n]^T$ and $\mathbf{d} = [d_1, \cdots, d_n]^T$. To this end, we update the weights of NN_C using the back propagation algorithm with momentum

$$\Delta w(l) = -\eta\frac{\partial\epsilon(l)}{\partial w} + \alpha\Delta w(l-1) \tag{2.4}$$

where w stands for any of the weights in Fig. 2, and $\eta > 0$ and $0 < \alpha < 1$ are the so-called learning and momentum constants, respectively. In order to use (2.4), one needs to determine $\frac{\partial\epsilon}{\partial w}$ explicitly considering the effect of w through K stages. Define $\mathbf{u}$ as $\mathbf{u} \triangleq [\mathbf{x}^T, \mathbf{o}^T]^T = [x_1, \cdots, x_n, o_1, \cdots, o_m]^T$ and further let

$$\begin{aligned}\mathbf{o}(k) &= C(\mathbf{x}(k)) = [C_1(\mathbf{x}(k)), \cdots, C_m(\mathbf{x}(k))]^T,\\ \mathbf{x}(k+1) &= D(\mathbf{u}(k)) = [D_1(\mathbf{u}(k)), \cdots, D_n(\mathbf{u}(k))]^T\end{aligned}$$

For simplicity of notation, denote partial derivatives of $C(\cdot)$ and $D(\cdot)$ as

$$C_{ij}(k) \triangleq \frac{\partial C_i(\mathbf{x}(k))}{\partial x_j(k))},\ D_{ij}(k) \triangleq \frac{\partial D_i(\mathbf{u}(k))}{\partial u_j(k))}$$

and collect these partial derivatives in two matrices, C' and D' as follows:

$$C' \triangleq \begin{bmatrix} \mathbf{I_n} \\ \hline C_{11} & \cdots & C_{1n} \\ \vdots & & \vdots \\ C_{m1} & \cdots & C_{mn} \end{bmatrix}, D' \triangleq \begin{bmatrix} D_{11} & \cdots & D_{1(n+m)} \\ \vdots & & \vdots \\ D_{n1} & \cdots & D_{n(n+m)} \end{bmatrix} \tag{2.5}$$

For the neural network structure given in Fig. 2, the C_{ij} terms in (2.5) can be computed as

$$C_{ij} = \dot{\varphi}(v_i)\sum_{l=1}^{N} w_{il}^o\dot{\varphi}(z_l)w_{lj}^h$$

For the weights in the output layer, we have

$$\frac{\partial\epsilon}{\partial w_{ij}^o} = (\mathbf{x}(K) - \mathbf{d})^T\frac{\partial\mathbf{x}(K)}{\partial w_{ij}^o} \tag{2.6}$$

In order to evaluate $\frac{\partial\mathbf{x}(K)}{\partial w_{ij}^o}$ in (2.6), we first derive a recursive equation for $k = 1, 2, \ldots, K$:

$$\frac{\partial\mathbf{x}(k)}{\partial w_{ij}^o} = D'(k-1)\frac{\partial\mathbf{u}(k-1)}{\partial w_{ij}^o} \tag{2.7}$$

$$\frac{\partial\mathbf{u}(k)}{\partial w_{ij}^o} = C'(k)\frac{\partial\mathbf{x}(k)}{\partial w_{ij}^o} + \dot{\Phi}_i(k)y_j(k) \tag{2.8}$$

where $\dot{\Phi}_i(k) \triangleq [0_{n+i-1}^T,\ \dot{\varphi}(v_i(k)),\ 0_{m-i}^T]^T$ and 0_j denotes the zero-vector of length j. From (2.7) and (2.8), we obtain

$$\frac{\partial\mathbf{x}(k)}{\partial w_{ij}^o} = \sum_{p=0}^{k-1} F'(k-1)\cdots F'(p+1)D'(p)\dot{\Phi}_i(p)y_j(p) \tag{2.9}$$

where $F'(i) \triangleq D'(i)C'(i)$, $i = 1, 2, \ldots$. Hence substituting (2.9) in (2.6) leads to

$$\frac{\partial\epsilon}{\partial w_{ij}^o} = (\mathbf{x}(K) - \mathbf{d})^T\sum_{p=0}^{K-1} F'(K-1)\cdots F'(p+1)D'(p)\dot{\Phi}_i(p)y_j(p)$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, N$. Similarly, for $i = 1, \ldots, m$:

$$\frac{\partial\epsilon}{\partial w_{bi}^o} = (\mathbf{x}(K) - \mathbf{d})^T\sum_{p=0}^{K-1} F'(K-1)\cdots F'(p+1)D'(p)\dot{\Phi}_i(p)$$

Proceeding in the same way for the weights in the hidden layer to derive a recursive equation for $k = 1, 2, \ldots, K$, we obtain

$$\frac{\partial\mathbf{x}(k)}{\partial w_{ij}^h} = D'(k-1)\frac{\partial\mathbf{u}(k-1)}{\partial w_{ij}^h} \tag{2.10}$$

$$\frac{\partial\mathbf{u}(k)}{\partial w_{ij}^h} = C'(k)\frac{\partial\mathbf{x}(k)}{\partial w_{ij}^h} + \Psi_i(k)\dot{\varphi}(z_i(k))x_j(k) \tag{2.11}$$

where $\Psi_i(k) \triangleq [0, \cdots, 0, \ \dot{\varphi}(v_1(k))w^o_{1i}, \cdots, \dot{\varphi}(v_m(k))w^o_{mi}]^T$. From (2.10) and (2.11), we obtain a closed-form expression similar to (2.9) as follows

$$\frac{\partial \mathbf{x}(k)}{\partial w^h_{ij}} = \sum_{p=0}^{k-1} F'(k-1)\cdots F'(p+1)D'(p)\Psi_i(p)\dot{\varphi}(z_i(p))x_j(p) \quad (2.12)$$

Using (2.12), one can compute $\frac{\partial \epsilon}{\partial w^h_{ij}}$ as

$$\frac{\partial \epsilon}{\partial w^h_{ij}} = (\mathbf{x}(K)-\mathbf{d})^T \sum_{p=0}^{K-1} F'(K-1)\cdots F'(p+1)D'(p)\Psi_i(p)\dot{\varphi}(z_i(p))x_j(p) \quad (2.13)$$

for $i = 1, \ldots, N$ and $j = 1, \ldots, n$. Similarly, for $i = 1, \ldots, N$, we have

$$\frac{\partial \epsilon}{\partial w^h_{0i}} = (\mathbf{x}(K)-\mathbf{d})^T \sum_{p=0}^{K-1} F'(K-1)\cdots F'(p+1)D'(p)\Psi_i(p)\dot{\varphi}(z_i(p)) \quad (2.14)$$

Having Equations (2.10), (2.11), (2.13), and (2.14) to determine partial derivatives of the final error with respect to the weights of the neural network NN_C, we can apply the back propagation algorithm in (2.4) to train NN_C.

In the remainder of this paper, the multi-stage neural network approach is applied to motion planning of a mechanical snake.

III Motion Planning of a Mechanical Snake

The basic design of a mechanical snake can be done in two steps: Design of its structure and design of its motion pattern. There are various studies on the design of flexible-articulated structures. These structures are designed either to obtain a more efficient robot arm or to realize the idea of producing snake-like mobile robots [3, 5].

Motion pattern designs, on the other hand, are usually based on the motion patterns of biological snakes, which can be divided into four groups: *Lateral undulation, rectilinear locomotion, sidewinding,* and *concertina motion.* Detailed explanations of these motion patterns of biological snakes are given in [6, 7]. Although lateral undulation is the most common one for snakes, concertina motion seems to be the most suitable method of motion to emulate. In concertina motion, the snake draws itself into an S-shaped curve and sets the curved portion of its body in static contact with the ground. Movement begins when the front part of the snake is extended by forces transmitted to the ground in the zone which remains in stationary contact. After the moving the head forward a short distance and establishing a new stationary contact zone in this new position, the rear end of the snake is pulled forward. In this motion pattern, the necessary force is produced by only contacting to the ground. Irregularities to exert force laterally are not needed. This fact and its suitability for progressing in narrow areas make the concertina motion appropriate for mechanical implementation and practical usage, and we base our design on this motion pattern. In our design, similar to [3], the mechanical snake proceeds by fixing some of its links and drawing S-shapes with free link sequences or straightening them successively.

III.1 Definition of the Problem

In this paper, we work with a structure that is composed of eight links of the same length. For each link, a link-solenoid is assumed to be able to fix the entire link to the surface, which can be realized using a solenoid pair for each link. For motion planning purposes, the structure can be illustrated as in Fig. 3(a). The joints connecting the links are denoted by J_i, $i = 1, \ldots, 7$ while J_0 and J_8 denote, respectively, the head and the tail points of the snake. We denote the ray $\overrightarrow{J_{i-1}J_i}$ by L_i for $i = 1, \ldots, 8$. The angles $\theta_0, \ldots, \theta_7$ are defined as follows. θ_0 is the directional counterclockwise (CCW) angle from x-axis to L_1 and θ_i is the directional (CCW) angle from L_i to L_{i+1} for $i = 1, \ldots, 7$. Similar to that in [3], $t_i \in \{0, 1\}$ denotes the state of the solenoid of Segment i for $i = 1, \ldots, 8$, where $t_i = 0$ denotes that the segment is free and $t_i = 1$ denotes that it is fixed by its solenoid. To avoid possible needs to apply large amounts of torque to swing big portions of the mechanical snake and to avoid a jack-knifed situation, in which the snake is bent over an intermediate joint and motion control is very difficult, absolute values of the interior joint angles θ_2, ..., θ_6 are limited to 60° with a tolerance of 5%.

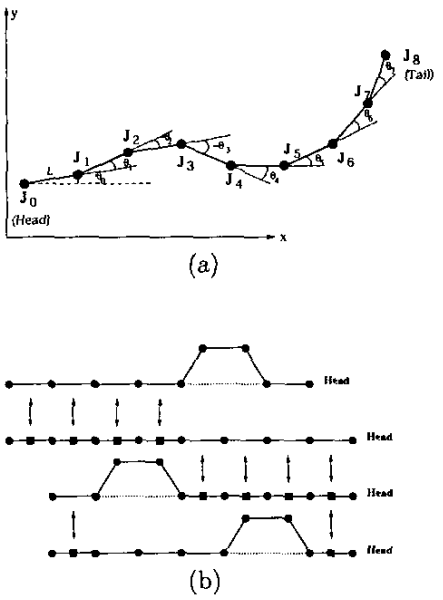

Figure 3: (a) Simplified structure of the mechanical snake. (b) Basic linear motion pattern.

Based on the structure and the angle constraint described above, the basic linear motion pattern is summarized in Table 1 and is also depicted in Fig. 3(b).

This pattern can be generalized for an arbitrary motion which does not violate the constraints by slightly modifying the joint angle changes (the activation of the solenoids is not affected). To simplify the general motion pattern, we adapt a convention that the sequence formed by the three links which are not on the path to be tracked (e.g. the line tracked for the basic linear motion pattern) is symmetric, i.e., the angles of the two joints which are not on the path are the same.

Step	t_1	t_2 t_3 t_4	t_5 t_6 t_7	t_8	θ_1	θ_2 θ_3	θ_4	θ_5 θ_6	θ_7
1	0	0	1	1	0°	0°	0°	0°	0°
2	1	1	0	0	0°	0°	−60°	60°	−60°
3	1	0	0	1	−60°	60°	−60°	0°	0°

Table 1: Segment solenoids and joint angles during a linear motion cycle.

Using the structure and the motion pattern described above, the task is to get the mechanical snake out of a rectangular region through a specified exit. The initial configuration of the snake is assumed to be suitable for following a smooth path to get out of the region without crashing on the borders. With this assumption, the explicit task to be dealt with is defined as follows.

Problem 1 *Consider a closed rectangular region R with corners at $(-1,-10)$, $(-1,10)$, $(20,-10)$ and $(20,10)$, and a single exit between the points $(-1,-2)$ and $(-1,2)$ (see Fig. 4(a)). Let the mechanical snake with unit length segments be initially located inside the square R_i with corners $(0,-10)$, $(0,10)$, $(20,-10)$ and $(20,10)$. Design a controller to take the snake out of R using the specified exit without violating the angle constraints $|\theta_i| \leq 63°$, $i = 2,\ldots,6$.*

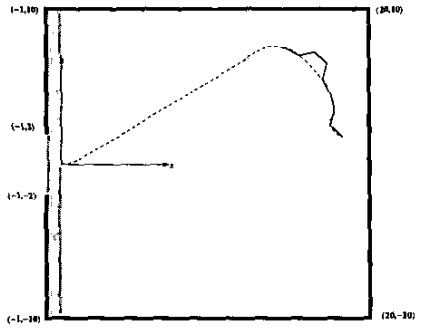

Figure 4: The rectangular region in Problem 1.

This task can be done in two steps: Generation of the path to get out, and guiding the mechanical snake to track the generated path. In this paper, path generation is performed using the multi-stage neural network approach discussed in Section 2. Once the path is generated, it can be tracked using an efficient algorithm (see, e.g., [7]).

III.2 Path Generation

To have an unobtrusive motion, and to take steps significant in size, the generated path should be a smooth one with no sharp turns, i.e., it should have a maximum curvature κ_{max}. A specific value for κ_{max} is determined from the following constraints:

C1. The interior joint angles are bounded by 63°, i.e., $|\theta_i| \leq 63°$ for $i = 2,3,4,5,6$.

C2. $|J_iJ_{i+3}| \geq 2.8$ if J_i, J_{i+1}, J_{i+2}, J_{i+3} are all lying on the path to be tracked.

C3. $|J_iJ_{i+3}| \leq 2.2$ if J_{i+1} and J_{i+2} are not lying on the path to be tracked.

Constraint C1 is imposed to avoid not only a jack-knifed position but the need to apply large amounts of torque to swing big portions of the mechanical snake as well. Constraints C2 and C3 ensure that the steps of the motion pattern produce significant movement. Using the constraints C1-C3, the maximum curvature is computed as $\kappa_{max} = 0.447$ [7]. Note that the algorithms and the neural network structures to be presented are independent of this specific value of κ_{max}, and the algorithms can be run and the neural networks can be trained for different maximum curvatures by only setting κ_{max} accordingly.

In the sequel we assume that the initial configuration is adjusted so that the snake can be represented by a sequence of seven pieces of curves, each of which is either a line segment of length 1 or an arc of radius $r^* \triangleq \frac{1}{\kappa_{max}}$ and chord length 1[1]. For such a configuration, five of the joints together with the head and the tail points coincide with seven of the end points of these curve pieces. Let us order the pieces as Piece 1, ..., Piece 7 from head to tail, and denote the end points of these pieces as V_0, ..., V_7 in the same order. Let P_i denote both the ray $\overrightarrow{V_{i-1}V_i}$ and the type of Piece i, $i = 1,\ldots,7$, where the piece-type definitions are given in Fig. 5(a). The directional (CCW) angle from x-axis to the ray which is tangent to Piece 1 at V_0 and which follows the direction of Piece 1 is denoted by ϕ_0, and the directional (CCW) angle from P_i to P_{i+1} is denoted by ϕ_i, $i = 1,\ldots,6$. Note that the convention used in defining ϕ_0 is different from the convention used in defining ϕ_i, $i = 1,\ldots,6$, and θ_i, $i = 0,\ldots,7$. Using this notation, a configuration is represented by 10 parameters, position and direction of the head, and types of the seven pieces $(x_{J_0}, y_{J_0}, \theta_0, P_1, \ldots, P_7)$. A sample initial configuration, which is represented by $(x, y, 180°, -1, -1, 1, -1, 0, 1, 1)$, is given in Fig. 5(b).

[1]A neural network to make this adjustment for an arbitrary initial condition is designed as well [7]. However presentation of this design is omitted in this paper.

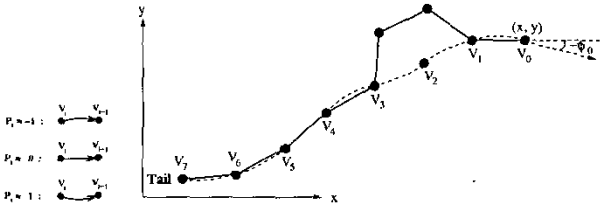

Figure 5: (a) Piece–type definitions.
(b) The configuration represented by $(x, y, 180°, -1, -1, 1, -1, 0, 1, 1)$.

In the path generation task, the goal can be simplified to find a path in R_i ending at the origin with a direction perpendicular to the exit interval. After that point, linear motion is sufficient to take the snake out. In the realization of the task, the path generated does not exactly end at the origin with the specified direction, but these values are close enough to the desired ones to exit from the specified interval.

The path generation problem can be reformulated as finding a path $X(t) = (x(t), y(t))$ in the right half plane such that $X(0) = X_0$, $\phi(0) = \phi_0$, $X(t_f) = 0$ and $\phi(t_f) = 180°$ for some t_f, and $\kappa(t) \leq \kappa_{max}$, $\forall t \in [0,\ t_f]$. Here, $\phi(t) = \text{atan2}(\dot{y}(t), \dot{x}(t))$, where atan2$(\cdot,\cdot)$ is the signed arctan function and $\kappa(t')$ is the curvature of the path $X(t)$ at $t = t'$. This problem is solved using multi–stage neural networks in the next section.

IV Design of the Path Generator

Based on the mechanical snake configuration introduced in Section III, we have designed a neural network that generates paths as compositions of the three piece-types defined in Fig. 5, with smooth connections. The inputs of the path generator neural network NN_{path} are $x = x_{J_0}$, $y = y_{J_0}$, $\theta = \phi_0$, and the output P' is the type of the next curve piece to be tracked (i.e., -1,0, or 1), the P_1 for the next step. NN_{path} is composed of a 3–input, 1–output, 2–layer neural network NN_C, which has $N = 100$ hidden neurons, and a 3–state hardlimiter, which quantizes the output of NN_C to produce one of the three piece-types.

NN_C is trained using the multi–stage neural network approach explained in Section II. The stopping criterion is chosen as the occurrence of either reaching a point with $x \leq 0$ or proceeding for 50 steps. The input–output relation of the path generator neural network is described by $\mathbf{o} = C(\mathbf{x})$, where $C(\mathbf{x})$ is given as

$$C(\mathbf{x}) = \varphi\left(w_{b1}^o + \sum_{i=1}^{N} w_{1i}^o \varphi\left(w_{bi}^h + \sum_{j=1}^{3} w_{ij}^h x_j\right)\right)$$

The components of the matrix C' in (2.5) can be evaluated as $C_{1j} = \dot{\varphi}(v_1)\sum_{i=1}^{N} w_{1i}^o \dot{\varphi}(z_i) w_{ij}^h$, $j = 1, 2, 3$.

The effect of the 3–state hardlimiter is taken into account while evaluating C', and in this evaluation, the hardlimiter is approximated by the smooth function $\frac{1}{2}(\varphi_D(x - 0.3) + \varphi_D(x + 0.3))$, where $\varphi_D(x) = \tanh(\frac{\lambda_D}{2}x)$, $\lambda_D = 100$.

Next, we outline how we find a smooth function $D(\cdot)$ to implement the dynamics giving the next position and direction of the snake's head, given the current values, and the output of the path generator, which indicates the type of the curve piece to be tracked. Various cases to be considered are shown in Fig. 6. Given $\mathbf{x} = [x,\ y,\ \theta]^T$, the function $D(\cdot)$ should satisfy the relations

$$D\left(\begin{bmatrix} \mathbf{x} \\ 0 \end{bmatrix}\right) = \begin{bmatrix} x + \cos\theta \\ y + \sin\theta \\ \theta \end{bmatrix} \tag{4.1}$$

$$D\left(\begin{bmatrix} \mathbf{x} \\ -1 \end{bmatrix}\right) = \begin{bmatrix} x + r^*(\sin\theta - \sin(\theta - \theta_\Delta)) \\ y + r^*(\cos(\theta - \theta_\Delta) - \cos\theta) \\ (\theta - \theta_\Delta)_{2\pi} \end{bmatrix} \tag{4.2}$$

$$D\left(\begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}\right) = \begin{bmatrix} x + r^*(\sin(\theta + \theta_\Delta) - \sin\theta) \\ y + r^*(\cos\theta - \cos(\theta + \theta_\Delta)) \\ (\theta + \theta_\Delta)_{2\pi} \end{bmatrix} \tag{4.3}$$

where $\theta_\Delta \triangleq 2\arcsin\left(\frac{1}{2r^*}\right) = 2\arcsin(0.5\kappa_{max})$.

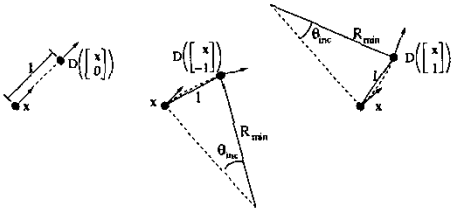

Figure 6: $D(\cdot)$ for the path generation problem.

One possible candidate function for $D(\cdot)$ satisfying these conditions is given by

$$D\left(\begin{bmatrix} \mathbf{x} \\ o \end{bmatrix}\right) = \begin{bmatrix} x + r^* o(s_{\theta + o\theta_\Delta} - s_\theta) + c_{o\frac{\pi}{2}} c_\theta \\ y + r^* o(c_\theta - c_{\theta + o\theta_\Delta}) + c_{o\frac{\pi}{2}} s_\theta \\ (\theta + o\theta_\Delta)_{2\pi} \end{bmatrix} \tag{4.4}$$

where c_ϕ denotes $\cos\phi$ and s_ϕ denotes $\sin\phi$. For smoothness, instead of the hardlimiter of this neural network, the sharp sigmoid $\varphi_D(x)$ defined above is used in the emulator, i.e., $D(\cdot)$ in (4.4) is implemented as

$$D\left(\begin{bmatrix} \mathbf{x} \\ o \end{bmatrix}\right) = \begin{bmatrix} x + r^* o(s_{\theta + o\theta_\Delta} - s_\theta) + c_{o\frac{\pi}{2}} c_\theta \\ y + r^* o(c_\theta - c_{\theta + o\theta_\Delta}) + c_{o\frac{\pi}{2}} s_\theta \\ \theta + o\theta_\Delta - \pi - \pi\varphi_D(\theta + o\theta_\Delta - 2\pi) \end{bmatrix} \tag{4.5}$$

From (4.5), the entries of the matrix D' in (2.5) can be derived as $D_{11} = D_{22} = 1$, $D_{12} = D_{21} = D_{31} = D_{32} = 0$, $D_{13} = r^* o(c_{\theta + o\theta_\Delta} - c_\theta) - c_{o\frac{\pi}{2}} s_\theta$, $D_{14} = r^*(s_{\theta + o\theta_\Delta} - s_\theta + o\theta_\Delta c_{\theta + o\theta_\Delta}) - \frac{\pi}{2} s_{o\frac{\pi}{2}} c_\theta$, $D_{23} = r^* o(s\theta + o\theta_\Delta - s_\theta) + c_{o\frac{\pi}{2}} c_\theta$, $D_{24} = r^*(c_\theta - c_{\theta + o\theta_\Delta} + o\theta_\Delta s_{\theta + o\theta_\Delta}) - \frac{\pi}{2} s_{o\frac{\pi}{2}} s_\theta$, $D_{33} = 1 - \pi\varphi'_D(\theta + o\theta_\Delta - 2\pi)$, $D_{34} = \theta_\Delta - \theta_\Delta \pi\varphi'_D(\theta + o\theta_\Delta - 2\pi)$.

V Numerical Results

We have constructed a training set of 200 (x, y, θ) data points chosen uniformly from the set $0 < x \leq 4$, $-2 \leq y \leq 2$, $\frac{3\pi}{4} \leq \theta \leq \frac{5\pi}{4}$. The neural network is trained for

5,000 epoches (one epoch is one back propagation cycle over the whole training set) using constant values for the learning parameters η and α. The time history of the average E (over one epoch) of the error function ϵ is plotted in Fig. 7.

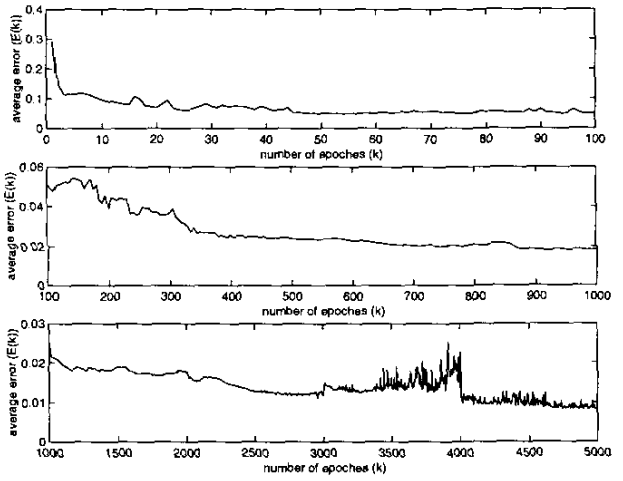

Figure 7: Average error during the training process.

We have also checked the path generation performance of the neural network for the weights of epoches 100, 1000, 5000. The outputs obtained using these weights in the neural network for three sample initial triples are shown in Fig. 8. As can be seen from the results after epoch 5000, the path generator neural network is successful enough in the entire domain of interest. The performance of the neural network can be further improved by adjusting the learning parameters η and α adaptively during the training process.

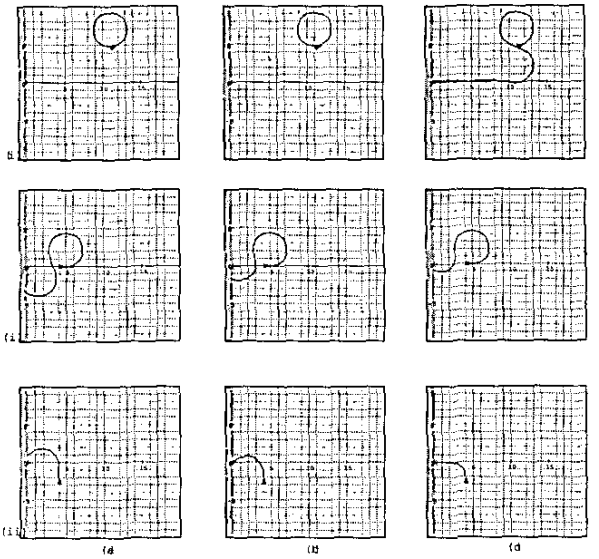

Figure 8: Path generation after (a)100, (b)1000 and (c)5000 epoches with $\mathbf{x}(0) =$ (i)$[12, 5, 30°]^T$, (ii)$[5, 0, 0°]^T$, (iii)$[4, -2, 90°]^T$.

VI Conclusion

In this paper, we have discussed the use of multi–stage neural networks in controlling general discrete-time systems and applied this approach to motion planning of a mechanical snake. A 2–layer neural network has been designed via multi–stage neural network approach for path generation. This neural network, together with a tracking algorithm has been used to move the mechanical snake out of a collision–free closed region successfully.

The performance of the neural network designed in this paper can be further improved by two different methods: Enlarging the domain of the training set elements as the training process progresses, and determining the elements of the training set by moving K_b random steps back from the desired final state, where K_b is increased as training advances. For these methods, fixing constant learning parameters to be used in every step of the training process is not feasible since the multiplicative terms in the weight update equations accumulate as the number of stages in Fig. 1 increases. A topic of further investigation is the application of the multi–stage neural network approach using adaptive learning parameters. Another possible future research topic is extending the design procedure presented in this paper to cases with obstacles.

References

[1] D. H. Nguyen and B. Widrow, "Neural networks for self-learning control systems," *IEEE Control Systems Magazine*, pp. 18–23, April 1990.

[2] D. Nguyen and B. Widrow, "The truck backer-upper," in *International Neural Network Conference*, vol. 1, (Paris), pp. 399–407, July 1990.

[3] Y. Shan and Y. Koren, "Design and motion planning of a mechanical snake," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, no. 4, pp. 1091–1100, 1993.

[4] D. Psaltis, A. Sideris, and A. A. Yamamura, "A multilayered neural network controller," *IEEE Control Systems Magazine*, pp. 17–21, April 1988.

[5] G. S. Chirikjian and J. W. Burdick, "A hyper-redundant manipulator," *IEEE Robotics & Automation Magazine*, pp. 22–29, December 1994.

[6] C. Gans, "How snakes move," *Scientific American*, vol. 222, pp. 82–96, 1970.

[7] B. Fidan, "Motion planning of a mechanical snake using neural networks," Master's thesis, Bilkent University, Ankara, Turkey, July 1998.