Sequential Behavior and Learning in Evolved Dynamical Neural Networks

Brian M. Yamauchi
Dept. of Computer Engineering and Science
Case Western Reserve University
Cleveland, OH 44106
yamauchi@alpha.ces.cwru.edu

Randall D. Beer
Dept. of Computer Engineering and Science
Dept. of Biology
Case Western Reserve University
Cleveland, OH 44106
beer@alpha.ces.cwru.edu

Technical Report CES-93-25

To appear in *Adaptive Behavior*

Abstract

This paper explores the use of a real-valued modular genetic algorithm to evolve continuous-time recurrent neural networks capable of sequential behavior and learning. We evolve networks that can generate a fixed sequence of outputs in response to an external trigger occurring at varying intervals of time. We also evolve networks that can learn to generate one of a set of possible sequences based upon reinforcement from the environment. Finally, we utilize concepts from dynamical systems theory to understand the operation of some of these evolved networks. A novel feature of our approach is that we assume neither an a priori discretization of states or time nor an a priori learning algorithm that explicitly modifies network parameters during learning. Rather, we merely expose dynamical neural networks to tasks that require sequential behavior and learning and allow the genetic algorithm to evolve network dynamics capable of accomplishing these tasks.

1. Introduction

Much of the recent research on autonomous agents has focused on reactive behavior, in which an agent's actions are either largely or exclusively determined by its immediate situation. Reactivity ensures that an agent can respond quickly to unanticipated contingencies and take advantage of unanticipated opportunities as they are presented by its environment. Reactivity is thus a crucial capability for any agent operating in an unpredictable environment.

However, many tasks, such as landmark-based navigation and maze negotiation, require that an agent also be able to integrate perceptual information over time in order to determine the appropriate course of action. An illustrative metaphor that we will use throughout this paper is that of a rat running a maze. As a rat traverses a maze, it is presented with a series of choice points at variable intervals of time. Each time it reaches an intersection, it must choose to go either left or right. While each intersection may be partially distinguished by perceptual cues, such as visual appearance or smell, in general perceptual information will be insufficient to uniquely determine the appropriate action. Instead, the rat must integrate these partial perceptual cues with its previous decisions in that run in order to determine the most appropriate action. Furthermore, in order to learn its way through new mazes, a rat must also be capable of learning new decision sequences. We will refer to the problem of generating a fixed sequence of decisions in response to decision triggers from the environment as the *sequence generation task* and the problem of learning to generate one of a possible set of sequences based upon environmental reinforcement as the *sequence learning task.*

Our previous work has demonstrated that continuous-time recurrent neural networks can be evolved to control a variety of sensorimotor behaviors, such as chemotaxis and legged locomotion (Beer and Gallagher, 1992), where the continuous dynamics of these networks appear to be a significant advantage. However, it is not at all obvious that this same general approach can be applied to tasks, such as landmark-based navigation or maze negotiation, that have a fundamentally discrete and sequential component. It is also not obvious that dynamical neural networks are capable of learning without significant modification to their basic equations of state. Therefore, in this paper, we investigate the ability of a genetic algorithm to evolve continuous-time recurrent neural networks that can solve the sequence generation and learning tasks. In addition, we employ concepts from dynamical systems theory in order to understand the operation of some of the networks that we evolve.

The paper is organized as follows. The basic neural model and genetic algorithm are described in Section 2. Section 3 defines the sequence generation task that we studied and presents and analyzes the operation of several evolved solutions. Section 4 defines the sequence learning task that we studied and presents and analyzes several evolved solutions. In Section 5, we compare our approach to related work on reinforcement learning, the induction of regular grammars by recurrent neural networks, and the relationship between learning and evolution. Finally, Section 6 considers the implications of our work for understanding the role of learning in autonomous agents and sketches some directions for future research.

2. Methods

2.1 Dynamical Neural Networks

In the experiments described in this paper, we employed continuous-time recurrent neural networks whose behavior is governed by a system of differential equations of the following form:

$$\tau_i \dot{y}_i = -y_i + \sum_{j=1}^{N} w_{ji}\sigma(y_j - \theta_j) + \sum_{k=1}^{S} s_{ki} I_k \quad i = 1, 2, \ldots, N \tag{1}$$

where y is the state of the neuron, τ is the time constant of the neuron, N is the total number of neurons, w_{ji} gives the strength of the connection from the jth to the ith neuron, $\sigma(\xi) = (1 + e^{-\xi})^{-1}$ is the standard sigmoidal activation function, θ is a bias term, S is the number of sensory inputs, I_k is the output of the kth sensor, and s_{ki} is the strength of the connection from the kth sensor to the ith neuron. In the experiments presented here, every neuron received an input from every sensor.

The initial state of all neurons was set to zero and the system of equations was integrated using the forward euler method with a stepsize of 1. Time constants were in the range [1,70], biases were in the range [-1,1] and connection weights were in the range [-5,5].

2.2 The Genetic Algorithm

A real-valued genetic algorithm with a modular encoding scheme was used to search the parameter space of the above dynamical neural networks. Network parameters were encoded as a vector of real numbers, with the time constant, bias, sensor weights and incoming connection weights for each neuron represented as an indivisible unit for the purposes of crossover. This modular encoding is motivated by our belief that a neuron's intrinsic parameters and input weights form a useful building block that would not be preserved by a more conventional binary coding of parameters which allowed crossover to occur at arbitrary points.

A population of 1000 networks was maintained. The initial population was typically produced by randomly selecting a real value for each parameter of each network with a uniform probability distribution over the range of allowable values. In some experiments, the initial population was seeded with combinations of networks evolved in previous runs. The performance of each network was evaluated on multiple trials and these individual performance scores were then combined into an overall fitness score for that network by subtracting the baseline performance (the expected performance of a network that was simply guessing) from the average performance of that network across the trials:

$$fitness = \frac{1}{T}\sum_{i=1}^{T} perf_i - perf_{base} \qquad (2)$$

where T is the number of trials, $perf_i$ is the performance of the network on the ith trial and $perf_{base}$ is the baseline performance (usually 0.5 in the experiments described in this paper). Negative fitness scores were set to zero.

Once fitness scores were assigned to all individuals in the population, a new population was generated. Individuals were selected for reproduction with a probability proportional to their fitness. Crossover occurred with a 30% probability in all of the experiments described in this paper. If crossover did not occur, a single parent was randomly selected. If crossover did occur, two parents were randomly selected and a crossover point *x* was randomly chosen. The child then received the parameters for neurons 1 through *x* from its first parent and the parameters for neurons *x*+1 through *N* from its second parent. Regardless of whether or not crossover occurred, there was a small probability (ranging from 0.2% to 1% in the experiments described here) that any parameter of the child would be mutated. When mutation occurred, a new value was randomly chosen for the parameter to be mutated with a uniform probability distribution over the range of allowable values.

3. Sequence Generation Experiments

3.1 The Task

If we idealize the decision trigger from the environment as a binary input (e.g., an intersection in a maze has been reached) and the decision to be made as a binary choice (e.g., turn either left or right), then the sequential decision-making problem becomes the problem of generating a particular sequence of binary digits in response to a sequence of environmental triggers occurring at variable intervals of time. While these idealizations abstract over many other important aspects of problems such as landmark-based navigation or maze negotiation (e.g., integrating decision-making with the actual sensorimotor control problems posed by these tasks), they do form a necessary subset of the capabilities required by these tasks. If the sequence repeats with a period *n*, then we will refer to it as an *n*-bit sequence generation task. For example, using regular expression notation, a maze for which the path to the goal consisted of alternating left and right turns would pose a 2-bit sequence generation task of the form (01)* or (10)* depending upon whether 0 or 1 was the correct decision for the first intersection. Note that generating an *n*-bit sequence requires the equivalent of a finite state machine with *n* states.

Fully interconnected dynamical neural networks with a single binary input and a single continuous output were employed. The decision trigger input I_1 normally had a value of 0. At variable intervals of time, however, the environment signaled that a decision must be made by setting I_1 to 1 for 10 time steps. The time interval between triggers was drawn from a uniform probability distribution over the range [10,40] time steps. By convention, neuron 1 always served as the output of the network. For the duration of a trigger, the output of neuron 1 was compared to the corresponding bit in the desired sequence. The overall performance of a network on a given sequence was computed as:

$$perf = 1 - \frac{1}{B}\sum_{i=1}^{B}\frac{1}{end_i - start_i}\int_{start_i}^{end_i}\left|b_i - \sigma(y_1(t) - \theta_1)\right| dt \qquad (3)$$

where B is the total number of bits in the sequence, b_i is the value of the ith bit, $\sigma(y_1(t) - \theta_1)$ is the output of neuron 1 (the designated output neuron) at time t, and $start_\text{i}$ and end_i are the times of the beginning and end, respectively, of the ith trigger. Evaluation sequences typically consisted of repetitions of an n-bit kernel (e.g., 01). Note that the output of neuron 1 between triggers is irrelevant. Note also that this performance score takes on values between 0 and 1 because the integral is normalized to the duration of the trigger and the sum is normalized to the number of bits in the sequence.

3.2 Two-Bit Sequence Generation

3.2.1 Results

We ran two separate experiments on the 2-bit sequence generation problem (10)*. Five-neuron, fully interconnected networks with a total of 40 free parameters were employed. Networks were evaluated on a sequence consisting of 12 repetitions of the kernel and a mutation rate of 1% was utilized. After 1000 generations, both populations contained many different dynamical neural networks that solved this 2-bit sequence generation problem. We will focus on the best network from each population here.

The behavior of the first network, which we will refer to as SG1, is shown in Figure 1a. Only three of the five neurons are actually used. After transients have passed, the remaining two neurons assume fixed values and can therefore be replaced by constant inputs to the rest of the network. Recall that neuron 1 is the output neuron for the network. The first time a trigger occurs, the output of this neuron is 1, then 0, then 1, etc. In other words, SG1 is successfully generating the desired sequence (10)* in response to environmental triggers. Note that SG1 is capable of holding an output state indefinitely, even though it was only required to do so for a maximum of 40 time steps during evolution.

[Insert Figure 1 About Here]

As soon as one trigger ends, SG1 makes the transition to the state appropriate for the next trigger. This is a common strategy of the solutions that evolved. This strategy stems from the fact that the performance measure described in equation (3) involves the integral of the difference between the desired and actual outputs over the duration of the trigger. This performance will obviously be higher for networks that are already in the correct state when a trigger occurs than for those that must make the transition during the trigger itself.

The behavior of the best network from the second population, SG2, is shown in Figure 1b. Once again, only three of the five neurons were actually used. Note that the output of SG2 is quite similar to that of SG1 on the same trigger sequence. However, the activity of the other neurons is quite different. Furthermore, note that, unlike SG1, SG2 is incapable of holding the 1 output state indefinitely. While SG2 can hold the 1 output state for over 100 time steps after the last trigger (more than twice as long as it was required to do during evolution), it eventually reverts to the 0 output state even in the absence of a subsequent trigger.

3.2.2 SG1 Analysis

How are we to understand the operation of these evolved networks? Elsewhere (Beer, in press a; Beer, in press b), we have argued that the mathematical theory of dynamical systems provides the appropriate conceptual framework for answering such questions. Dynamical systems theory allows us to characterize the possible long-term behaviors of a system and the dependence of those behaviors on parameters. A complete review of the modern theory of dynamical systems is clearly beyond the scope of this paper. However, a number of good introductions to dynamical systems theory are now available (Wiggins, 1990; Hale and Koçak, 1991; Abraham and Shaw, 1992) and the interested reader should consult one of these sources for further details.

We will utilize only a few basic concepts from dynamical systems theory in this paper. A subset of a dynamical system's state space that attracts all nearby trajectories is called an *attractor*. If the attractor is a single point, then it is called an *equilibrium point attractor.* In terms of dynamical neural networks, an equilibrium point attractor corresponds to a fixed pattern of activity in all of the neurons that is stable to small perturbations. If, on the other hand, the attractor is a closed trajectory, it is called a *limit cycle*. A limit cycle corresponds to a time-varying pattern of activity in a dynamical neural network that repeats rhythmically. The set of all trajectories that converge to a given attractor is called its *basin of attraction*.

The general strategy of our analysis is to first understand how the dynamics of a network varies as a function of the sensory input that it receives and then to attempt to explain its operation in these terms. Here the decision trigger input serves as a binary parameter that switches the dynamics of SG1 between two distinct phase portraits. Because both SG1 and SG2 utilize only three of their five neurons, we can easily visualize these phase portraits in three-dimensional plots.

Figure 2 shows the three-dimensional output space projection of the phase portrait of network SG1 as the trigger switches between 0 and 1. Only stable limit sets are shown. When the trigger is off (Figures 2b and 2d), SG1 has two equilibrium point attractors (henceforth referred to as the *trigger-off* equilibrium points). If the trigger remains off for a sufficiently long period of time, the state of SG1 will always settle into one of these two attractors. Note that the locations of these attractors correspond to states in which the output of neuron 1 will be 0 or 1. On the other hand, when the trigger is on (Figures 2a and 2c), SG1 has only one equilibrium point attractor near the lower front right-hand corner (henceforth referred to as the *trigger-on* equilibrium point). Regardless of its initial state, the system will be attracted toward this equilibrium point as long as the trigger remains on.

[Insert Figure 2 About Here]

We can now understand the operation of SG1 as follows. Initially, let us assume that SG1 is in the 0 output state (i.e., its state is in the vicinity of the lower left-hand attractor in Figures 2b and 2d). When the trigger goes on, the trigger-on equilibrium point appears in the lower front right-hand corner and the system flows toward it (Figure 2a). After 10 time steps, the trigger goes off. Since the system is now in the basin of the upper trigger-off attractor (corresponding to an output state of 1), the system now flows toward this attractor (Figure 2b). After 10-40 time steps, the trigger goes on again for 10 time steps and the system is once again attracted to the trigger-on equilibrium point (Figure 2c). Note that, this time, the trigger goes off before the system actually reaches this equilibrium point. However, when the trigger does go off, the system state is now in the basin of the lower trigger-off attractor (corresponding to an output state of 0) and therefore flows back toward it (Figure 2d). Thus we can see how the trigger-on equilibrium point alternatively switches SG1 between the basins of attraction of the two trigger-off equilibrium points. Since each output state corresponds to a stable trigger-off equilibrium point of the network, we can also understand why SG1 can hold either output state indefinitely.

3.2.3 SG2 Analysis

The operation of SG2 is somewhat more complicated than SG1. Nevertheless, we can apply this same basic approach to analyzing the operation of SG2. Figure 3 shows the three-dimensional output space projection of the phase portrait of network SG2 as the trigger switches between 0 and 1. Once again, only stable limit sets are shown. When the trigger is off, SG2 exhibits a single equilibrium point attractor at a location corresponding to an output state of 0 (Figures 3b and 3d). Although it is not apparent from the figures, another important feature of the trigger-off phase portrait is that there is a region in the front upper left-hand corner of the state space where the vector field is very small. The system state will therefore move very slowly through this region of the state space. When the trigger is on, SG2 exhibits a single limit cycle attractor (Figures 3a and 3c).

[Insert Figure 3 About Here]

We can now understand the operation of SG2 as follows. Initially, let us assume that SG2 is in the vicinity of the trigger-off equilibrium point, a state corresponding to an output of 0. When the trigger goes on, the trigger-off equilibrium point disappears and the system state is attracted to the limit cycle (Figure 3a). After 10 time steps, the trigger goes off and the state now flows toward the trigger-off equilibrium point (Figure 3b). However, its path takes it through the slow region in the front upper left-hand corner of the state space, which corresponds to an output of 1. When the trigger goes on again, the slow region disappears and the state is attracted to the top of the limit cycle (Figure 3c). When the trigger goes off 10 time steps later, the state quickly flows toward the trigger-off equilibrium point because it is no longer in the slow region. Thus we can see how the limit cycle alternately switches SG2 between the trigger-off equilibrium point (whose location corresponds to an output of 0) and the slow region (whose location corresponds to an output of 1).

Note that this analysis explains why SG2 is incapable of holding an output state of 1 indefinitely. Unlike SG1, SG2 has no equilibrium point attractor corresponding to an output state of 1. If the trigger stays off long enough, the state will eventually flow to the single trigger-off equilibrium point. However, because the state moves so slowly through the slow region when the trigger is off, SG2 can hold an output state of 1 considerably longer than 40 time steps, the maximum intertrigger duration.

3.3 Three-Bit Sequence Generation

A 10-neuron network that solves the 3-bit task (110)* was also evolved using a population size of 1000 with a mutation rate of 1%. The behavior of the best network after 500 generations is shown in Figure 4. Only seven of the neurons are actually used by this circuit. Note that this network makes the transition to the 0 state only after holding the 1 state for two triggers, as is required by the (110)* task. Due to the larger number of neurons utilized by this network, we will not analyze its operation here.

[Insert Figure 4 About Here]

4. Sequence Learning Experiments

4.1 The Task

If the sequence generation task is analogous to evolving agents that can solve a particular maze, then the sequence learning task is analogous to evolving agents that can learn to solve different mazes based on environmental reinforcement. The sequence learning task that we studied was very similar to the sequence generation task described earlier except that, rather than generating a single fixed sequence in response to decision triggers from the environment, the network must be able to generate one of a set $\mathcal{B}$ of possible sequences based on environmental reinforcement. The reinforcement of a given decision was delayed by a variable amount of time, but always occurred before the next environmental trigger. Thus, while the network could not rely upon receiving reinforcement at a fixed point in time, we did not address the general problem of delayed reinforcement.

Fully interconnected dynamical neural networks with two inputs, one binary (the trigger) and one continuous (the reinforcement signal), and one continuous output (the decision) were employed. The trigger input and single output operated as before. Following a decision trigger, reinforcement was presented for a fixed duration of 10 time steps with a delay uniformly distributed over the range [10,40] time steps. During the i^{th} reinforcement, the magnitude of the reinforcement signal I_2 was computed according to the following formula:

$$I_2 = 1 - \frac{1}{end_i - start_i} \int_{start_i}^{end_i} \left| b_i - \sigma(y_1(t) - \theta_1) \right| dt \tag{4}$$

where b_i is the value of the i^{th} bit, $\sigma(y_1(t) - \theta_1)$ is the output of neuron 1 (the designated output neuron) at time t, and $start_i$ and end_i are the times of the beginning and end, respectively, of the

ith reinforcement. Note that a network whose output is the complement of b_i will receive a total reinforcement of 0, while a network whose output is b_i will receive a total reinforcement of 1. The interval between the presentation of the reinforcement signal for the network's response to one trigger and the next trigger was uniformly distributed over the range [10,40] time steps. The performance of a network for each member of the set $\mathcal{B}$ of possible sequences was evaluated as described by equation (3) for the sequence generation task. We will term a trial in which, say, the sequence (01)* was reinforced a (01)* *positive* trial. The performances of these individual trials were then combined as described in equation (2).

It is important to understand exactly what the genetic algorithm was being asked to do. Achieving a high fitness score on this task requires the ability to learn which of a set of possible sequences is the appropriate one to generate using environmental reinforcement. However, there was no explicit learning algorithm present in these networks and no mechanism for adjusting network parameters was available. Furthermore, the reinforcement was merely an additional sensory input that was not explicitly labeled as a reinforcement signal. Thus, the genetic algorithm was being asked to discover that the reinforcement sensor is in fact providing information about which output sequence is the appropriate one and then to evolve networks that could use this information to select the appropriate sequence to generate.

4.2 One-Bit Sequence Learning

4.2.1 Results

In our first learning experiment, we sought to evolve 5-neuron networks that could learn to generate the set of 1-bit sequences $\mathcal{B} = \{0^*, 1^*\}$ based on environmental reinforcement. Since these sequences correspond to constant outputs, the sequence generation problem to be solved is trivial. However, there is still an interesting learning problem to be solved because the network must utilize the reinforcement signal in order to determine which constant output state to adopt. Two experiments were run for 500 generations, with a mutation rate of 1%. Both experiments consisted of four trials.

In the first experiment, networks were evaluated on two 0* positive and two 1* positive trials. The behavior of one such network is shown in Figure 5. This network, which we will call SL1, initially adopts an output state of 0. If this output is reinforced, then it remains in this state as long as it is reinforced (Figure 5a). If, however, this output is not reinforced, then SL1 adopts an output of 1 and then remains in this state as long as it receives positive reinforcement (Figure 5b).

[Insert Figure 5 About Here]

We found that the networks that evolved in this experiment were in general only capable of one-time learning. The initial reinforcement that they receive can irreversibly determine their subsequent behavior. Once this initial period is over, these networks are not in general capable of changing their behavior even if the pattern of reinforcement later changes. For example, while the network shown in Figure 5 can make the transition from 0* to 1* if the pattern of reinforcement later changes from 0* positive to 1* positive, once it has been exposed to 1* positive reinforcement, this network is incapable of making the reverse transition from 1* to 0* regardless of the pattern of reinforcement it receives. While the underlying mechanisms are surely quite different, this behavior is reminiscent of what is sometimes called a “critical period”, where the subsequent development of an animal can be drastically and irreversibly altered by its experiences within a crucial period of time. We will examine the basis for this property in SL1 in Section 4.2.2.

In the second experiment, networks were evaluated on one 0* positive trial, one 1* positive trial, a 0* positive followed by a 1* positive trial, and a 1* positive followed by a 0* positive trial. The idea here was to explicitly encourage the evolution of networks that could modify their behavior whenever the pattern of reinforcement changed, rather than having their plasticity limited to an initial critical period. The behavior of one such network, which we will call SL2, is shown in Figure 6. The initial behavior of SL2 is very similar to that of SL1. SL2 first adopts an output state of 0 and remains in this state as long as it is reinforced. However, if this state is not reinforced, then SL2 shifts to an output state of 1 and remains there as long as it receives positive reinforcement. Unlike SL1, however, SL2 does indeed retain its plasticity throughout, since it is able to later make the transition both from 0* positive to 1* positive (Figure 6a) and from 1* positive to 0* positive (Figure 6b).

[Insert Figure 6 About Here]

4.2.2 SL1 Analysis

In principle, we can follow the same basic approach to dynamical analysis of SL1 and SL2 as we did for the sequence generation networks analyzed earlier. However, this strategy is complicated by a number of factors. First, SL1 and SL2 employ all five of their neurons rather than the three utilized by the sequence generation networks. Since we cannot directly visualize five-dimensional phase portraits, we will have to limit ourselves to three-dimensional projections. Second, while the sequence generation networks had only a single binary parameter (the trigger), the sequence learning networks also have a continuous parameter (the reinforcement signal) in addition to the binary trigger. For simplicity, our analysis will assume that the reinforcement signal is also binary, with low reinforcements treated as 0 and large reinforcements treated as 1. Finally, since SL1 and SL2 sometimes require multiple triggers and reinforcements before fully learning a desired sequence, the step-by-step analysis we performed on the sequence generation networks would be rather tedious to perform here. Therefore, we will settle for a more abbreviated presentation. With these caveats in mind, we now turn to the analysis of SL1.

Figure 7a-c shows how the three-dimensional output space projection of the phase portrait of SL1 varies as a function of the trigger and reinforcement signals. When both inputs are off (Figure 7a), SL1 exhibits two stable equilibrium points, one near the back, lower left-hand corner (corresponding to an output of 0) and one near the front, upper right-hand corner (corresponding to an output of 1). Henceforth, these will be referred to as the *input-off* equilibrium points. When the trigger is on (Figure 7b), SL1 exhibits a single equilibrium point attractor at the front, lower left-hand corner (henceforth referred to as the *trigger-on* equilibrium point). Finally, when the reinforcement signal is on (Figure 7c), SL1 exhibits a single equilibrium point attractor near the back, lower left-hand corner (henceforth referred to as the *reinforcement-on* equilibrium point).

[Insert Figure 7 About Here]

Given this information, we can understand the behavior of SL1 during learning as follows. Let us assume that SL1 is initially in the neighborhood of the lower input-off equilibrium point (Figure 7a), corresponding to an output state of 0. During a 0* positive trial, the state is pulled toward the trigger-on equilibrium point shown in Figure 7b whenever the trigger comes on. However, when the reinforcement signal comes on, the state is pulled toward the back, lower left-hand corner by the reinforcement-on equilibrium point shown in Figure 7c. When the reinforcement goes off, the state is once again attracted by the lower input-off equilibrium point. Thus, during a 0* positive trial, the state is alternately attracted by the lower input-off, trigger-on and reinforcement-on equilibrium points. This causes SL1 to move back and forth along the lower left-hand edge as shown in Figure 7d, an area corresponding to an output of 0.

During a 1* positive trial, on the other hand, SL1 receives no reinforcement during the first trigger because its initial output of 0 is incorrect. Since the reinforcement-on equilibrium point never has a chance to pull the system back into the neighborhood of the lower input-off equilibrium point, the system is attracted to the upper input-off equilibrium point when the trigger goes off. The location of this equilibrium point at the top of the state space corresponds to an output state of 1 (Figure 7e). Subsequent triggers and reinforcements serve only to slightly perturb SL1's state around this equilibrium point because the network's dynamics in this region is relatively insensitive to changes in the decision trigger and reinforcement inputs. This explains why SL1 can make the transition from 0* to 1* at any time, but is not capable of making the reverse transition after its "critical period" has passed.

4.2.3 SL2 Analysis

This same basic approach can be used to analyze the operation of SL2. Figure 8a-c shows how the three-dimensional output space projections of the phase portrait of SL2 varies as a function of the trigger and reinforcement signals. When both inputs are off (Figure 8a), SL2 exhibits a limit cycle. As we will see shortly, an important feature of this limit cycle is that one end passes through the lower region of the state space (corresponding to an output of 0), while the other end passes through the upper region (corresponding to an output of 1). When the trigger is on (Figure 8b), SL2 exhibits a single equilibrium point attractor near the front, lower right-hand corner of the state space. Finally, when the reinforcement signal is on (Figure 8c), SL2 exhibits a single equilibrium point attractor at the rear, lower right-hand corner of the state space.

[Insert Figure 8 About Here]

With this information in hand, we can understand the operation of SL2 as follows. We will assume that SL2 is initially in the neighborhood of the bottom right-hand end of the limit cycle, corresponding to an output of 0. If the network received no input whatsoever, it would follow this limit cycle, alternately producing outputs of 0 and 1. However, during a 0* positive trial, a trigger will occur after at most 40 time steps, causing the system state to be pulled toward the trigger-on equilibrium point shown in Figure 8b. After the trigger goes off, the state will again be pulled toward the limit cycle, but on a 0* positive trial, a reinforcement signal will occur within 40 time steps of the trigger. This causes the system to be pulled toward the reinforcement-on equilibrium point shown in Figure 8c. When the reinforcement goes off, the state will once again be pulled toward the limit cycle. Thus, during a 0* positive trial, the state of SL2 is pulled toward the limit cycle, but the equilibrium points that occur during the trigger and reinforcement signals keep pulling it back down. The resulting tug-of-war between the limit cycle and the equilibrium points serves to keep the system state in the lower right-hand edge of the state space, corresponding to an output of 0. Figure 8d shows a typical trajectory of a 0* positive trial.

During a 1* positive trial, the initial output of 0 is not reinforced. Since the reinforcement-on equilibrium point never has a chance to pull the system state back down, it travels up toward the top of the limit cycle, producing an output of 1. When subsequent triggers and reinforcements occur, they pull the state part of the way back down. As soon as these inputs turn off, however, the state will again be pulled toward the top of the limit cycle. This leads to a somewhat different tug-of-war between the limit cycle and the trigger-on and reinforcement-on equilibrium points which serves to keep the state in the vicinity of the upper left-hand corner of the output space, corresponding to an output of 1. A typical trajectory of a 1* positive trial is shown in Figure 8e.

The presence of the limit cycle also allows us to explain how, unlike SL1, SL2 retains its plasticity throughout its operation. Whenever the current output is not reinforced, the reinforcement-on equilibrium point never has a chance to pull the state back from the limit cycle, allowing the limit cycle to temporarily "win" the tug-of-war. If the state is near the bottom of the limit cycle when this occurs, then the limit cycle will pull it to the top. Alternately, if the state is near the top of the limit cycle, then the state will be pulled toward the bottom. Either way, an unreinforced output will cause SL2 to switch its behavior accordingly. Since there are only two possibilities in this 1-bit task, subsequent reinforcement will then hold the system in its new state.

4.3 Two-Bit Sequence Learning

In our second set of learning experiments, we sought to evolve dynamical neural networks that could learn to generate the set of sequences $\mathcal{B}$ = {0*, 1*, (01)*, (10)*} based on environmental reinforcement. Unfortunately, attempts to directly evolve such networks were unsuccessful. However, we were able to produce solutions to this problem using an incremental approach. First, we evolved 5-neuron networks that could learn to generate the more limited set of 2-bit sequences {(01)*, (10)*}. Of course, we already have 5-neuron networks that can learn to generate the sequences 0* and 1* from our previous experiments. Thus, we seeded a population with 10-neuron networks formed by fully interconnecting {0*, 1*} networks with {(01)*, (10)*} networks and evolved this population on the full 2-bit sequence learning task, with the output neuron of the {0*, 1*} network serving as the output neuron for the composite network. Two such experiments were run for 500 generations, with a mutation rate of 0.2% and $perf_{\text{base}}$ set to the average performance of each generation rather than a fixed value of 0.5.

The behavior of one solution to the full 2-bit learning task is shown in Figure 9. Note that this particular network made use of only nine of its neurons. Figure 9a shows this network on a (01)* positive trial followed by a 0* positive trial. After receiving one low reinforcement following its output of near 1 during the first trigger, the network locks into the (01)* sequence. When the pattern of reinforcement changes from (01)* positive to 0* positive after eight triggers, it takes the network three trigger-reinforcement pairs before it successfully makes the transition to 0*. Part of this delay is due to the fact that the network's output of 0 on the ninth trigger is fortuitously reinforced because 0* and (01)* share the same next bit. It is not until the tenth trigger that a low reinforcement following an output of 1 signals that a switch in behavior is necessary. Figure 9b shows the same network on a (10)* positive trial followed by a 1* positive trial.

[Insert Figure 9 About Here]

Recall that this network was evolved from the composition of two 5-neuron solutions to the {0*, 1*} and {(01)*, (10)*} learning tasks, respectively, with the output neuron of the {0*, 1*} subnetwork serving as the output neuron for the composite network. In light of this design, we examined the composition of the final network. We found that the parameters of the 5-neuron {(01)*, (10)*} subnetwork were virtually unchanged. Indeed, this subnetwork was still capable of solving the {(01)*, (10)*} learning task in isolation. However, the {0*, 1*} subnetwork had been completely redesigned in the course of evolving the full network and was no longer capable of solving the {0*, 1*} learning task on its own.

4.4 Three-Bit Sequence Learning

In our final set of learning experiments, we sought to evolve networks that could learn the set of 3-bit sequences $\mathcal{B}$ = {(011)*, (101)*, (110)*} based on environmental reinforcement. Note that this is a subset of the complete set of 3-bit sequences. All of the members of this particular subset are related by a phase shift. The 3-bit sequence (101)* is equivalent to (011)* after a 120° phase shift and (110)* is equivalent to (011)* after a 240° phase shift. A population of 10-neuron networks was evolved on this task for 700 generations with a mutation rate of 0.2%.

Figure 10 shows the behavior of a typical solution on a sequence of trials. Note that this particular solution utilizes only nine of its neurons. The sequence (011)* is first reinforced, followed by (101)*, and finally (110)*. The network successfully learns all three sequences.

[Insert Figure 10 About Here]

5. Related Work

The work described in this paper is perhaps most closely related to the large body of research on reinforcement learning algorithms. Kaelbling’s book (Kaelbling, 1993) provides an excellent introduction to and survey of the application of reinforcement learning to the problem of controlling an agent’s actions in some environment. Well-known reinforcement learning algorithms include Sutton’s temporal difference methods (Sutton, 1988) and Watkins’ Q-learning (Watkins, 1989). Variations of these reinforcement learning algorithms have been successfully applied to considerably more sophisticated problems than the simple sequence learning tasks considered here.

However, an important assumption of traditional reinforcement learning algorithms is that the behavior of the world can be explicitly decomposed into a set of discrete states and the essential problem to be solved is that of constructing a discrete map from situations to actions. In contrast, our approach does not assume the discretization of either world states or sensory inputs. Indeed, the experiments described in this paper demonstrate the ability of dynamical neural networks to generalize over similar temporally continuous patterns. While both Maes and Brooks (Maes and Brooks, 1990) and Chapman and Kaelbling (Chapman and Kaelbling, 1991) have investigated approaches that generalize over similar states by ignoring inputs that appear to be irrelevant, the degree to which individual inputs influence the generalization of responses in our approach is continuously variable. Millan and Torres (Millan and Torres, 1992) proposed a method for using reinforcement learning in domains where inputs and outputs are both continuous, but they considered only reactive behavior in which each input is mapped to a particular distribution of outputs. In contrast, our networks are capable of generating and learning simple state-dependent sequences of behavior. Another important difference between all of the traditional reinforcement learning approaches and our work is that we utilize no explicit parameter adjustment algorithm during learning.

Another related area of research concerns the induction of finite state machines by recurrent neural networks (Cleeremans, Servan-Schreiber et al., 1989; Pollack, 1991; Giles, Miller et al., 1992; Watrous and Kuhn, 1992). In this approach, recurrent neural networks are trained to recognize example strings from simple regular grammars. In many cases, such networks learn to mimic finite state machines that are closely related to the minimal FSM for the training grammar. However, while the inputs, outputs and states of these networks are continuous, time is not. Rather, discrete transitions between states are specified a priori by an external clock. In addition, like the work on reinforcement learning, this work utilizes explicit supervised training algorithms to adjust network parameters during learning.

Finally, there has been a small amount of work on the evolution of learning (Chalmers, 1991; Miller and Todd, 1991) and the relationship between learning and evolution (Hinton and Nowlan, 1987; Belew, McInerney et al., 1990). However, like the work on reinforcement learning, all of this work assumes some explicit algorithm for adjusting network parameters. In work on the relationship between learning and evolution, the learning algorithm (typically some form of backpropagation) is assumed to be fixed in advance, while in work on the evolution of learning, the parameters of the learning algorithm itself are evolved, but the form of the algorithm and its parameterization are still fixed. This work has also focused on using static feedforward networks to learn input/output maps rather than problems involving internal state.

6. Discussion

In this paper, we have demonstrated that small (3-9 neuron) continuous-time recurrent neural networks are capable of solving simple sequence generation and learning tasks. Furthermore, we have demonstrated that such networks can be evolved using a real-valued genetic algorithm with a modular encoding of network parameters. In addition, we have utilized basic concepts from dynamical systems theory to understand the operation of some of these evolved networks.

While the results described in this paper are rather modest as compared to the state-of-the-art in reinforcement learning and finite state machine induction, we believe that the significance of our results lies not so much in the complexity of the tasks we have considered, but in the minimalist approach that we have taken. Unlike approaches based on finite state machines or discrete-time recurrent neural networks, we have assumed no intrinsic discretization of states or time. More importantly, unlike approaches based on reinforcement learning or supervised learning techniques, we have assumed no explicit learning algorithm that modifies network parameters during behavior. Instead, we merely exposed continuous-time recurrent neural networks to tasks that require the generation and learning of environmentally-triggered sequences of discrete decisions and allowed the genetic algorithm to evolve network dynamics that accomplished these tasks. Here, learning merely consists in adopting a new, more appropriate dynamic mode of interaction between an agent and its environment rather than some long-term structural change to the agent itself.

It is interesting to note that, despite the fact that current models of learning almost universally assume an a priori discretization of states and time and an explicit algorithm for modifying system parameters during learning, no convincing evidence for such properties has ever been found in nervous systems. Our results suggest that, although such assumptions may be convenient for some purposes, they are fundamentally unnecessary. We have demonstrated that sequential decision making and learning can arise directly from the dynamics of continuous-time recurrent neural networks and, whatever else they may be, nervous systems are certainly continuous-time dynamical systems. Thus, our approach may eventually lead to more neurobiologically plausible models of learning. Furthermore, the fact that we were able to reliably evolve dynamical neural networks capable of sequential behavior and learning suggests that these are not especially rare properties of the dynamics of these networks.

More fundamentally, our results suggest a much tighter integration between behavior and learning in autonomous agents than is usually assumed. Models of learning typically draw a very sharp distinction between the mechanisms responsible for an agent's behavior and those responsible for learning. Indeed, some might go so far as to argue that our networks are not really learning precisely because they exhibit no such distinction. However, this distinction is difficult to defend biologically, because many of the same biochemical processes are involved in both processes. Likewise, our networks merely exhibit dynamics across a range of time scales suited to the time scales of the tasks we have set. This suggests that, in a continuous dynamical system, a rigid distinction between behavior and learning may be inappropriate and the more relevant distinction may be simply that between different time scales of dynamics in the same system.

A final observation that we would like to make is that, unlike most papers on learning, we have offered no general learning algorithm that can, for example, learn to generate arbitrary decision sequences or hold arbitrary states for arbitrarily long periods of time. Rather, the networks that we have presented evolve just enough plasticity to accomplish the particular tasks we have set for them. Unsurprisingly, the genetic algorithm took whatever shortcuts it could in the design of these networks, including holding states only as long as necessary and being able to learn to make only the transitions between sequences that it was required to do during evolution. Some might argue that this special-purpose nature is a severe limitation of our approach, but we think otherwise. Truly general-purpose learning is extremely difficult and there is little evidence that most animals are capable of such a feat. Instead, animals exhibit a multitude of special-purpose learning capabilities that are often exquisitely tuned to the particular niche that they occupy. For example, while rats can learn to associate illness with a particular odor or taste, they generally cannot learn to make a similar association with auditory or visual stimuli (Garcia, Hankins and Rusiniak, 1974). Likewise, we believe that the best way to incorporate learning into autonomous agents is by selectively integrating the necessary plasticity directly into the mechanisms responsible for particular behaviors.

We would like to point out that our approach has robotic applications as well. Dynamical neural networks evolved in simulation have been successfully applied to predator avoidance and landmark recognition in a Nomad 200 mobile robot equipped with sonar rangefinders

(Yamauchi, 1993). The avoidance task consisted of evading a moving pursuer while avoiding collisions with stationary obstacles. The landmark recognition task consisted of identifying different landmarks based on continuous sonar range data obtained as the robot circled the landmarks.

In conclusion, we believe that the ability of continuous-time recurrent neural networks to integrate perception and internal state in continuous domains on a variety of time scales and to transform this information into appropriate actions offers an interesting new approach to sequential behavior and learning in autonomous agents. Our future work will focus on exploring how far this approach can be pushed. In particular, we are interested in tackling the full landmark-based navigation problem, including the problem of learning the appropriate sequences of landmarks in novel environments. This will require some major extensions to our approach, including the ability to integrate sequential behavior and learning into more sophisticated sensorimotor behaviors and the ability to cope with more significant delays in reinforcement.

Acknowledgments

We would like to thank Hillel Chiel for his comments on an earlier draft of this paper. This work was supported in part by grant N00014-90-J-1545 from the Office of Naval Research. The robot experiments were performed at the Navy Center for Applied Research in Artificial Intelligence at the Naval Research Laboratory.

References

Abraham, R. H. and C. D. Shaw (1992). *Dynamics - The Geometry of Behavior, Second Edition*. Redwood City, CA: Addison-Wesley.

Beer, R. D. (in press a). A dynamical systems perspective on agent-environment interaction. To appear in *Artificial Intelligence.*

Beer, R. D. (in press b). Computational and dynamical languages for autonomous agents. To appear in T. van Gelder and R. Port, eds., *Mind as Motion*. Cambridge, Mass.: MIT Press.

Beer, R. D. and J. C. Gallagher (1992). Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior* **1**: 91-122.

Belew, R., J. McInerney and N. N. Schraudolph (1990). Evolving networks: Using the genetic algorithm with connectionist learning. CSE Technical Report CS90-174, University of California, San Diego.

Chalmers, D. J. (1991). The evolution of learning: An experiment in genetic connectionism. In D. S. Touretzky, J. L. Elman, T. J. Sejnowski and G. E. Hinton, eds., *Connectionist Models: Proceedings of the 1990 Summer School* (pp. 81-90). San Mateo, CA: Morgan Kaufmann.

Chapman, D. and L. P. Kaelbling (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings of the International Joint Conference on Artificial Intelligence* [IJCAI-91] (pp. 726-731), Morgan Kaufmann.

Cleeremans, A., D. Servan-Schreiber and J. L. McClelland (1989). Finite state automata and simple recurrent networks. *Neural Computation* **1**(3): 372-381.

Garcia, J., W.G. Hankins and K.W. Rusiniak (1974). Behavioral regulation of the milieu interne in man and rat. *Science* **185**:824-831.

Giles, C. L., C. B. Miller, D. Chen, H. H. Chen, G. Z. Sun and Y. C. Lee (1992). Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation* **4**(3): 393-405.

Hale, J. K. and H. Koçak (1991). *Dynamics and Bifurcations*. New York: Springer-Verlag.

Hinton, G. E. and S. J. Nowlan (1987). How learning can guide evolution. *Complex Systems* **1**: 495-502.

Kaelbling, L. P. (1993). *Learning in Embedded Systems.* Cambridge, MA: MIT Press.

Maes, P. and R. Brooks (1990). Learning to coordinate behaviors. In *Proceedings of the Eighth National Conf. on AI* (pp. 896-802), San Mateo, CA: Morgan Kaufmann.

Millan, J. del. and C. Torres (1992). A reinforcement connectionist approach to robot path finding in non-maze-like environments. *Machine Learning* **8**: 363-395.

Miller, G. F. and P. M. Todd (1991). Exploring adaptive agency I: Theory and methods for simulating the evolution of learning. In D. S. Touretzky, J. L. Elman, T. J. Sejnowski and G. E. Hinton, ed., *Connectionist Models: Proceedings of the 1990 Summer School* (pp. 65-80). San Mateo, CA: Morgan Kaufmann.

Pollack, J. B. (1991). The induction of dynamical recognizers. *Machine Learning* **7**: 227-252.

Sutton, R. S. (1988). Learning to predict by the method of temporal differences. *Machine Learning* **3**(1): 9-44.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. Phd Thesis, University of Cambridge.

Watrous, R. L. and G. M. Kuhn (1992). Induction of finite-state languages using second-order recurrent networks. *Neural Computation* **4**(3): 406-414.

Wiggins, S. (1990). *Introduction to Applied Nonlinear Dynamical Systems and Chaos*. New York: Springer-Verlag.

Yamauchi, B. M. (1993). Dynamical neural networks for mobile robot control. NRL Memorandum Report, Naval Research Laboratory, Washington, D.C.

Figure Captions

Figure 1: Behavior of sequence generation networks SG1 (a) and SG2 (b) on the 2-bit sequence (01)*. The vertical scale of all neuron activity plots in this and subsequent figures is [0,1] and, by convention, neuron 1 is always the output neuron. The desired sequence is shown immediately below the output neuron for reference.

Figure 2: Dynamical analysis of SG1. Only the stable limit sets are shown. Equilibrium point attractors are shown as black dots and system trajectories are shown as gray lines. Because neuron 1 is always the output neuron, the system state being anywhere near the top of this cube corresponds to an output state of 1 and anywhere near the bottom corresponds to an output state of 0. (a) Trigger on for 10 time steps. (b) Trigger off for 40 time steps. (c) Trigger on for 10 time steps. (d) Trigger off for 40 time steps.

Figure 3: Dynamical analysis of SG2. Only the stable limit sets are shown. Equilibrium points are shown as black dots, limit cycles are shown as black lines and system trajectories are shown as gray lines. (a) Trigger on for 10 time steps. (b) Trigger off for 40 time steps. (c) Trigger on for 10 time steps. (d) Trigger off for 40 time steps.

Figure 4: Behavior of a typical 3-bit sequence generation network on the sequence (110)*.

Figure 5: Behavior of the sequence learning network SL1 on (a) a 0* positive trial and (b) a 1* positive trial.

Figure 6: Behavior of the sequence learning network SL2 on (a) a 0* positive to 1* positive transition and (b) a 1* positive to 0* positive transition.

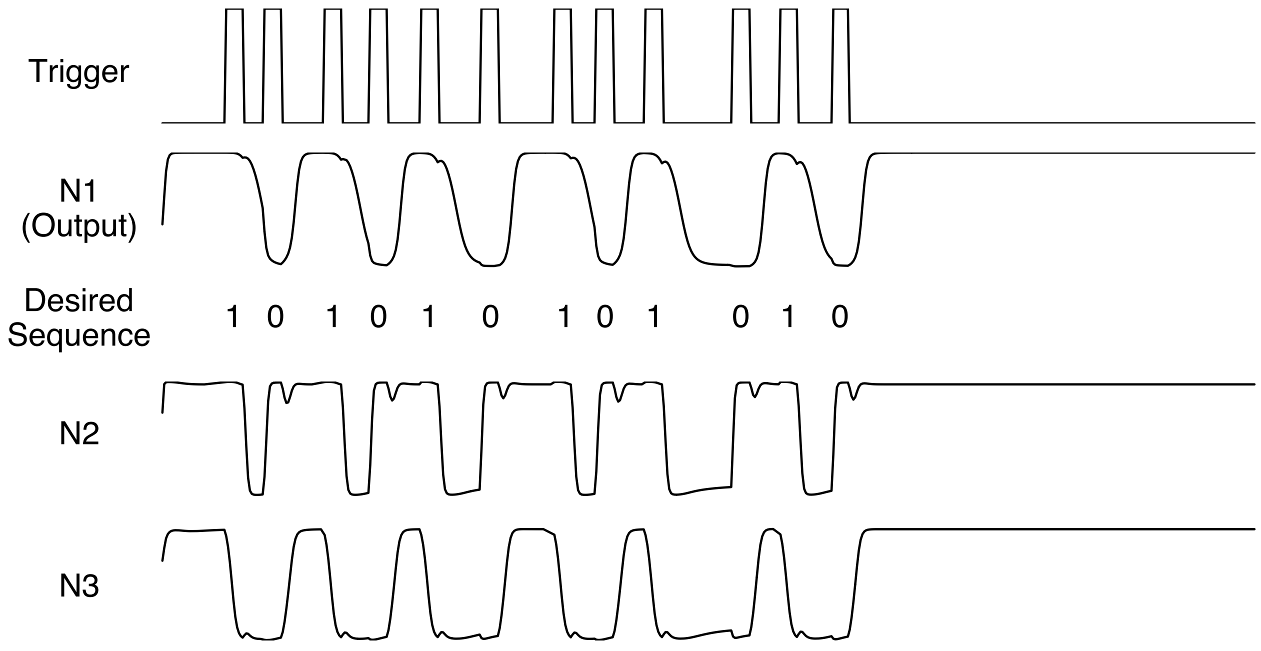
Figure 7: Dynamical analysis of SL1. Plots a-c shown the equilibrium point attractors of SL1 when (a) both the trigger and reinforcement signals are off, (b) when the trigger is on, and (c) when the reinforcement signal is on. (d) The trajectory of SL1 during a typical 0* positive trial. (e) The trajectory of SL1 during a typical 1* positive trial.

Figure 8: Dynamical analysis of SL2. Plots a-c shown the stable limit sets of SL2 when (a) both the trigger and reinforcement signals are off, (b) when the trigger is on, and (c) when the reinforcement signal is on. (d) The trajectory of SL2 during a typical 0* positive trial. (e) The trajectory of SL2 during a typical 1* positive trial.

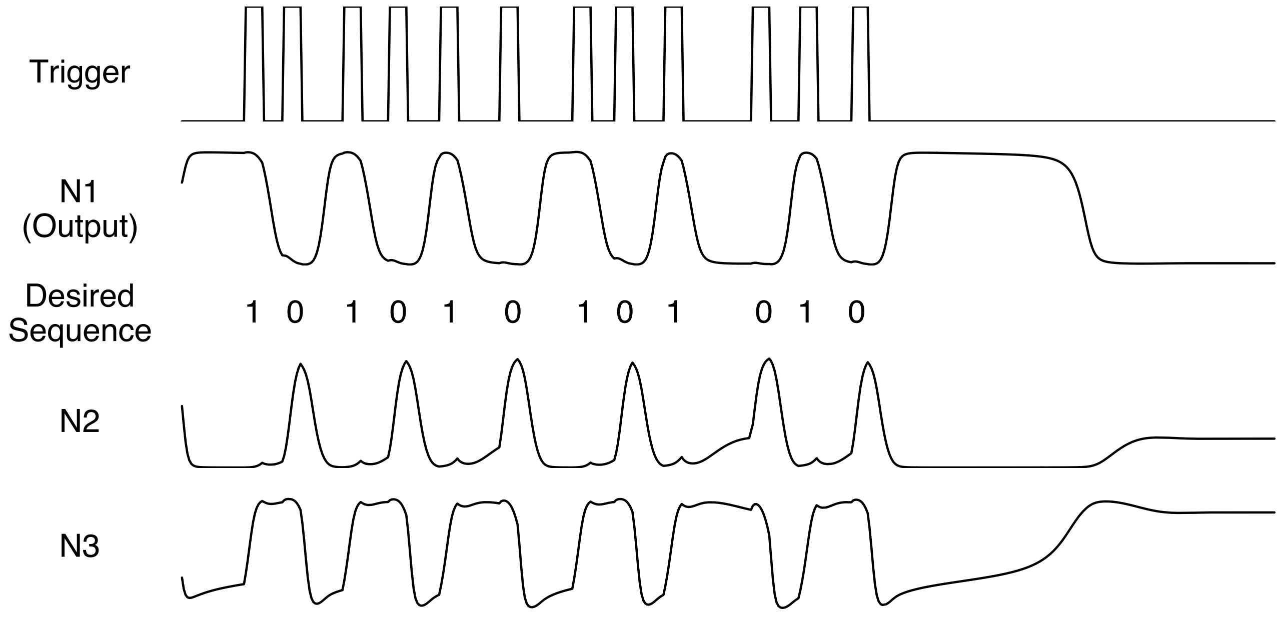
Figure 9: Behavior of a typical 2-bit sequence learning network. (a) A (01)* positive trial followed by a 0* positive trial. (b) A (10)* positive trial followed by a 1* positive trial.

Figure 10: Behavior of a typical 3-bit sequence learning network learning to generate first the sequence (011)*, then the sequence (101)*, and finally the sequence (110)*.

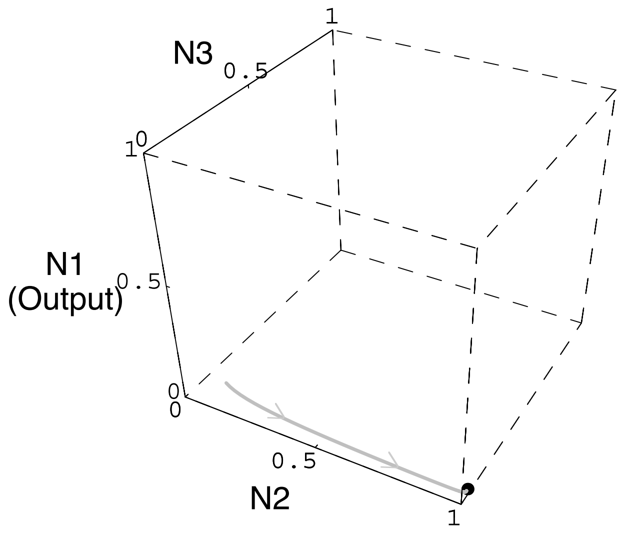
A


Trigger
N1
(Output)
Desired
Sequence
1 0 1 0 1 0 1 0 1 0 1 0
N2
N3

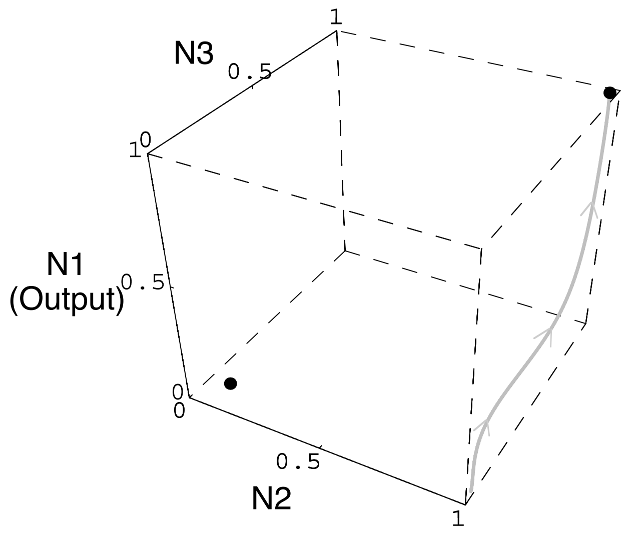
B


Trigger
N1
(Output)
Desired
Sequence
1 0 1 0 1 0 1 0 1 0 1 0
N2
N3

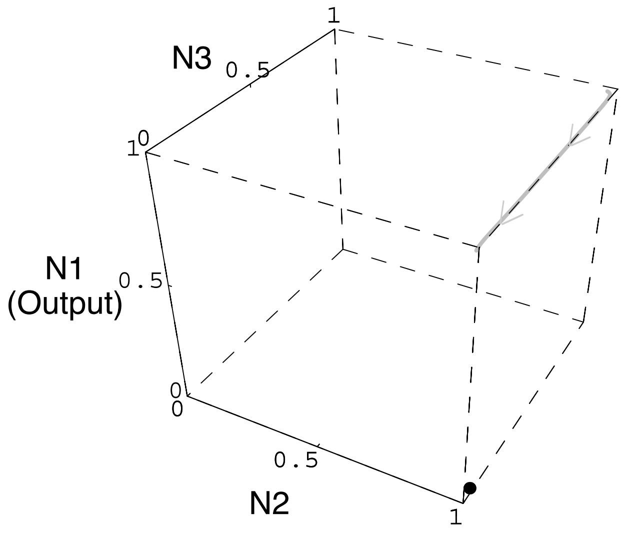
A


N3
0.5
1
0
1
N1
(Output)
0.5
0
0
0.5
N2
1

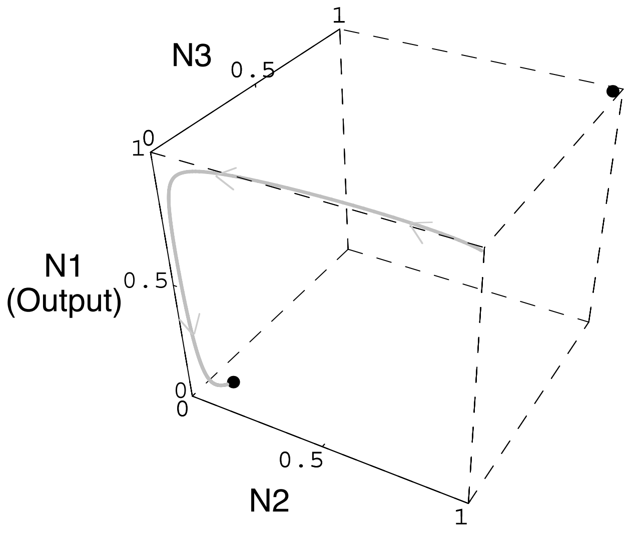
B


N3
0.5
1
0
1
N1
(Output)
0.5
0
0
0.5
N2
1

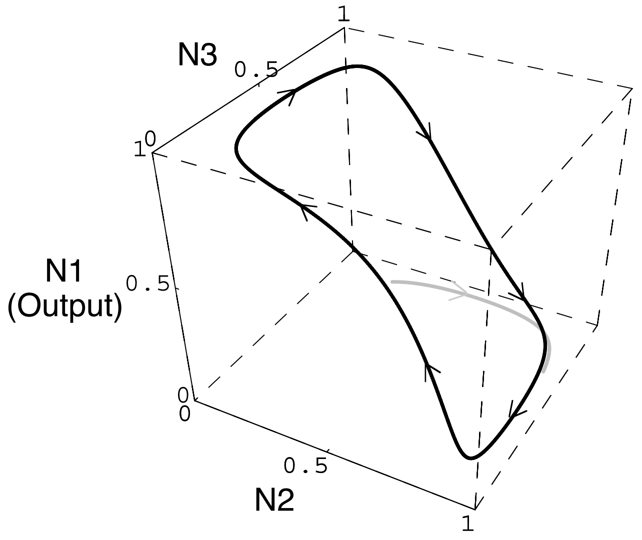
C


N3
0.5
1
0
1
N1
(Output)
0.5
0
0
0.5
N2
1

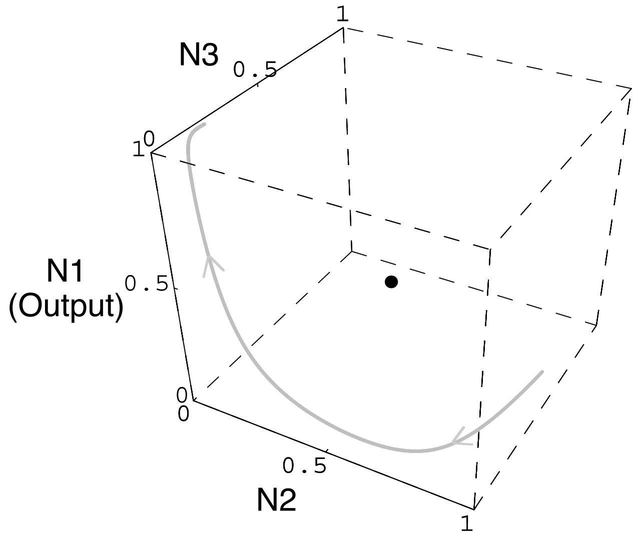
D


N3
0.5
1
0
1
N1
(Output)
0.5
0
0
0.5
N2
1

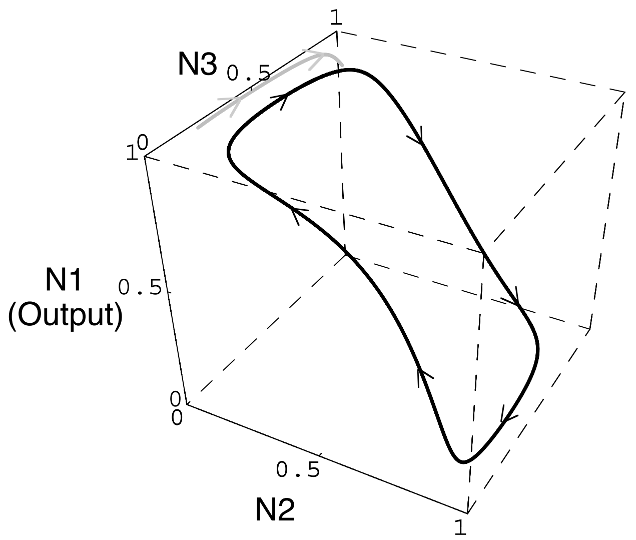
A


1
N3
0.5
1
0
N1
(Output)
0.5
0
0
0.5
N2
1

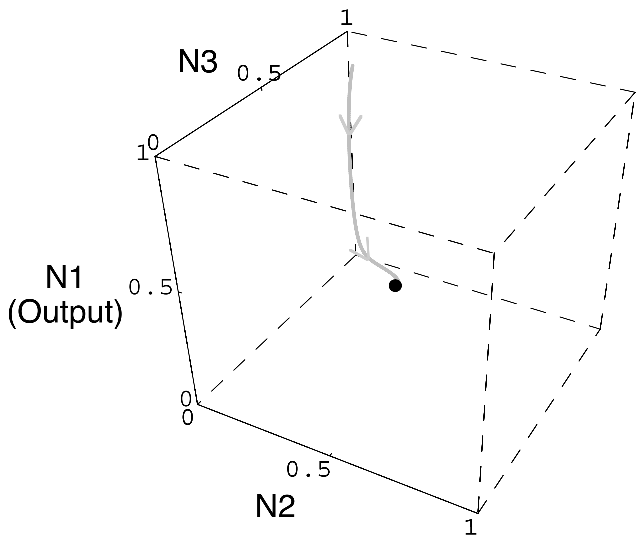
B

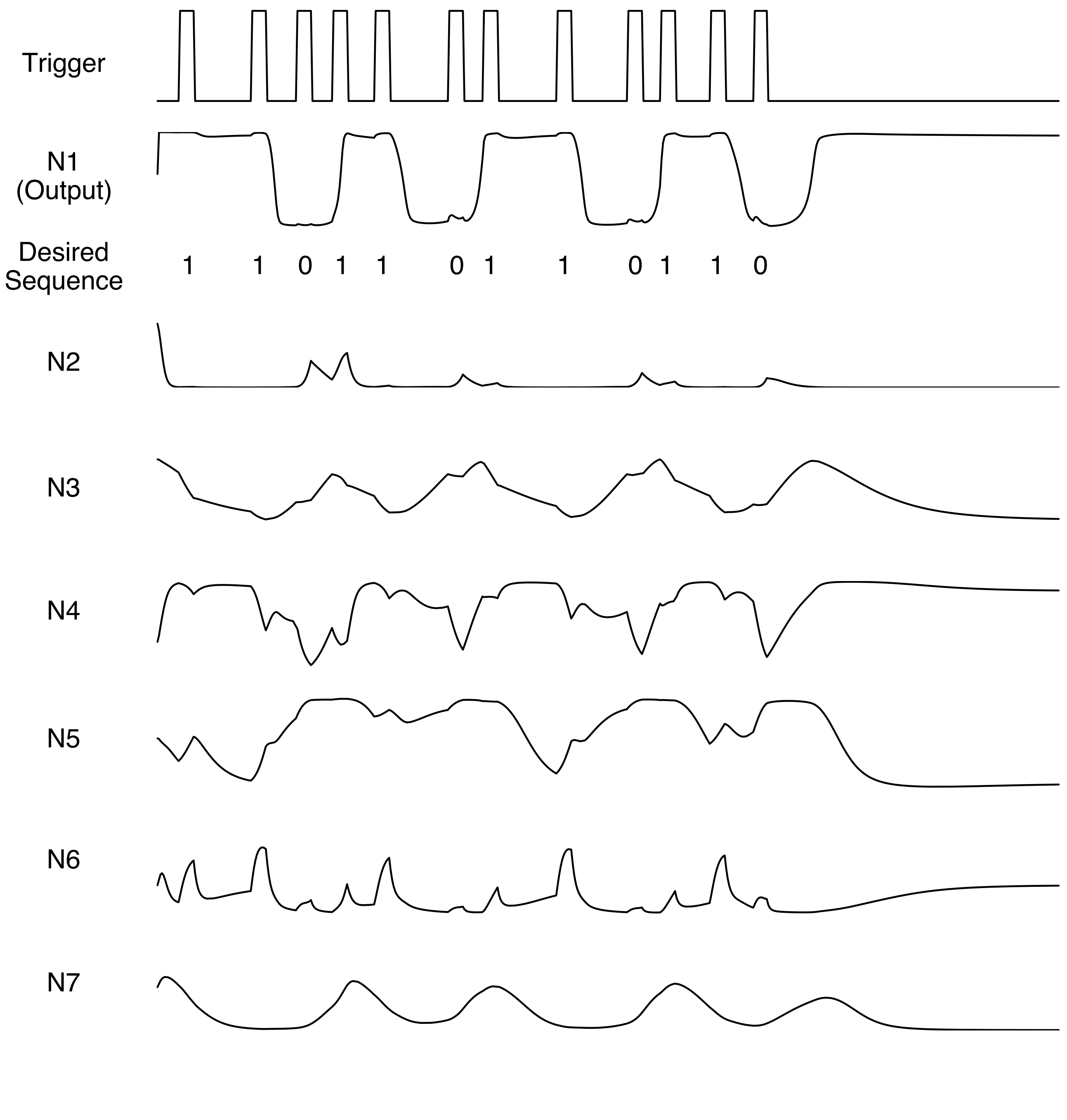

1
N3
0.5
1
0
N1
(Output)
0.5
0
0
0.5
N2
1

C

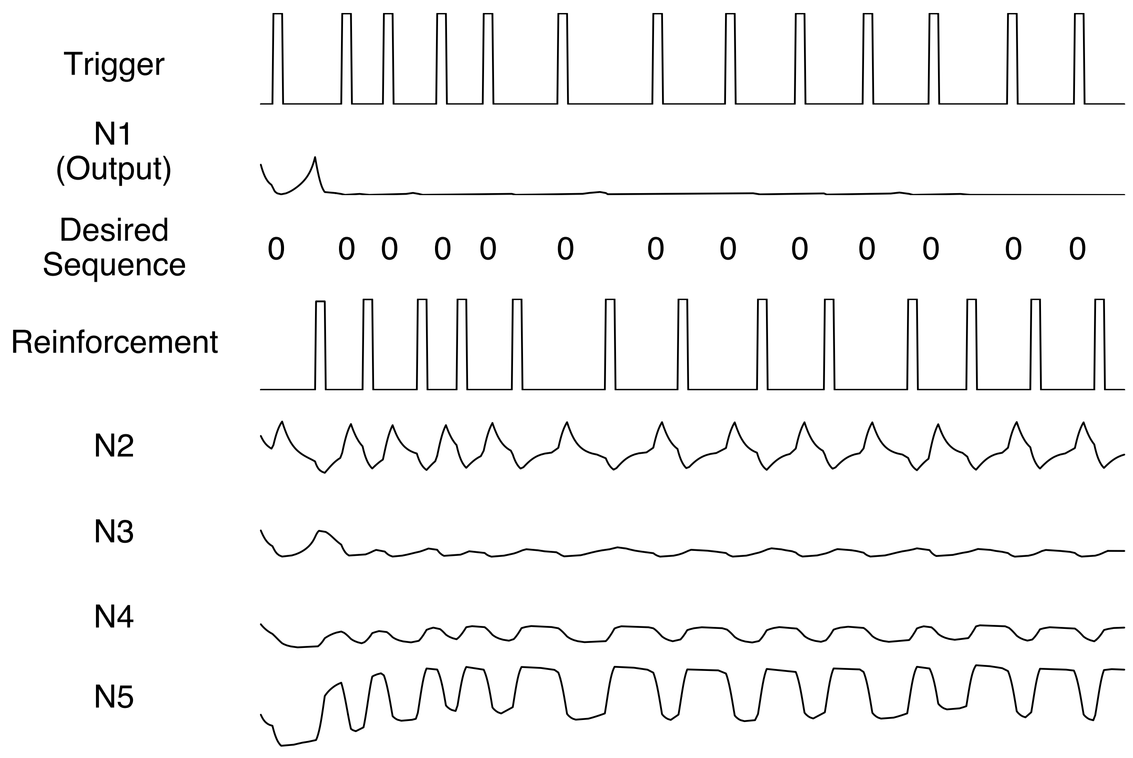

1
N3
0.5
1
0
N1
(Output)
0.5
0
0
0.5
N2
1

D

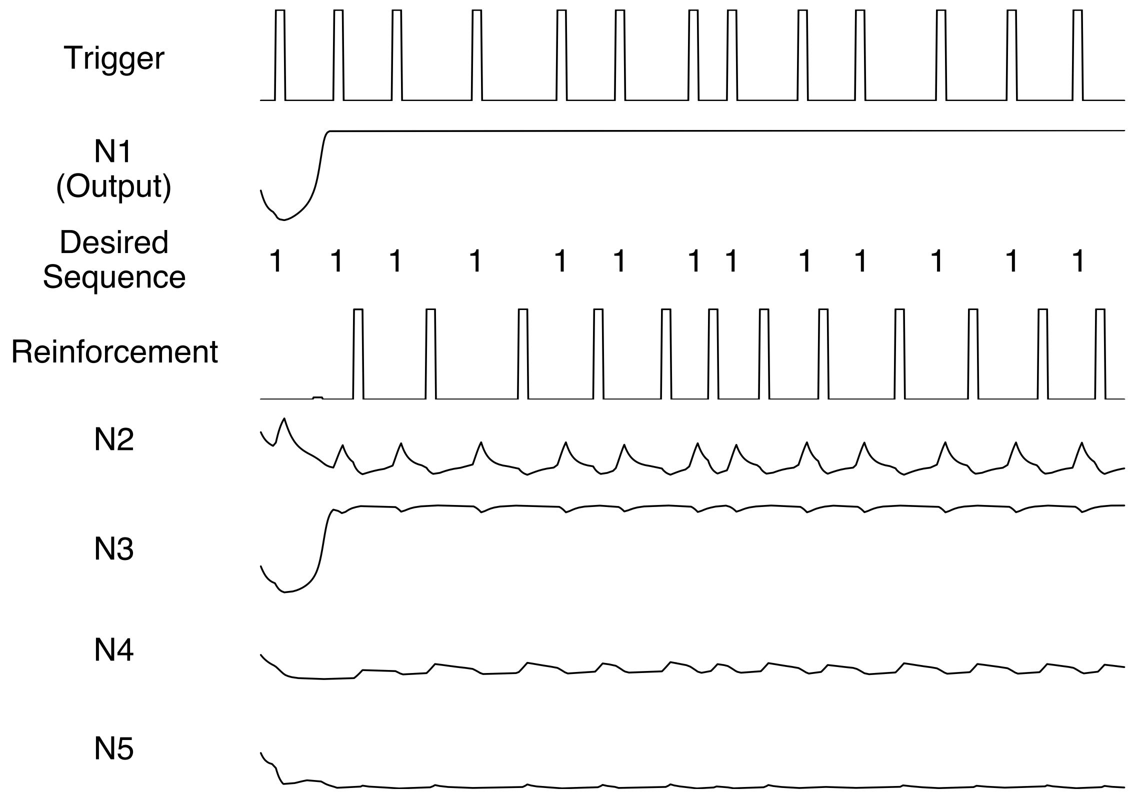

1
N3
0.5
1
0
N1
(Output)
0.5
0
0
0.5
N2
1


Trigger
N1
(Output)
Desired
Sequence
1 1 0 1 1 0 1 1 0 1 1 0
N2
N3
N4
N5
N6
N7

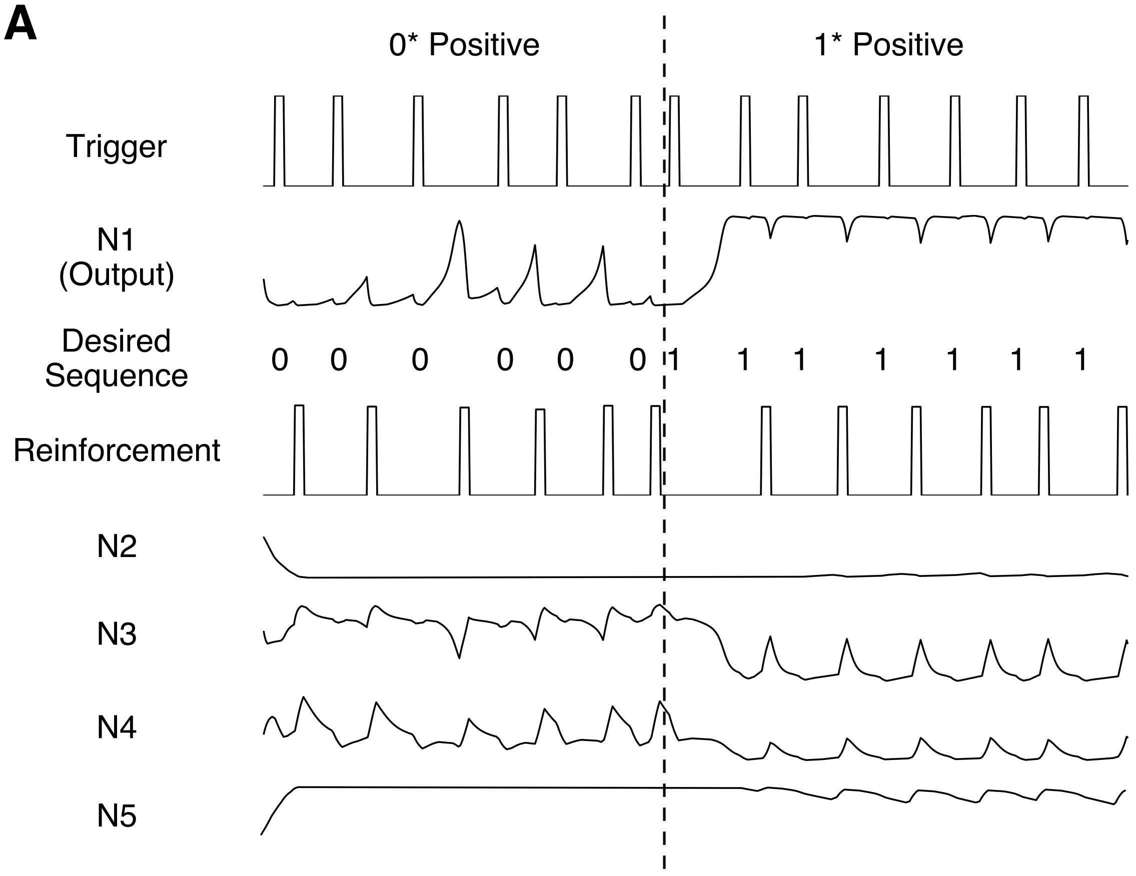
A


Trigger
N1
(Output)
Desired
Sequence
0 0 0 0 0 0 0 0 0 0 0 0 0
Reinforcement
N2
N3
N4
N5

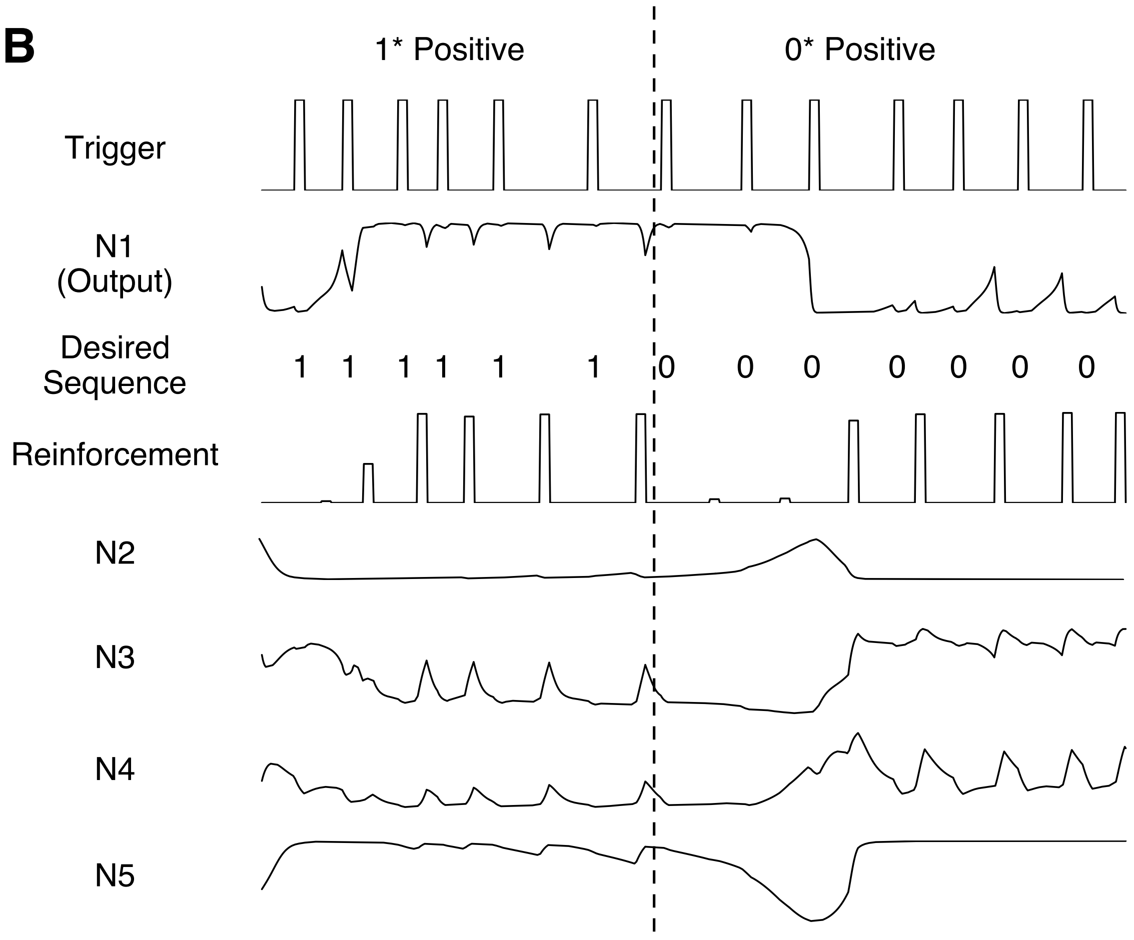
B


Trigger
N1
(Output)
Desired
Sequence
1 1 1 1 1 1 1 1 1 1 1 1 1
Reinforcement
N2
N3
N4
N5

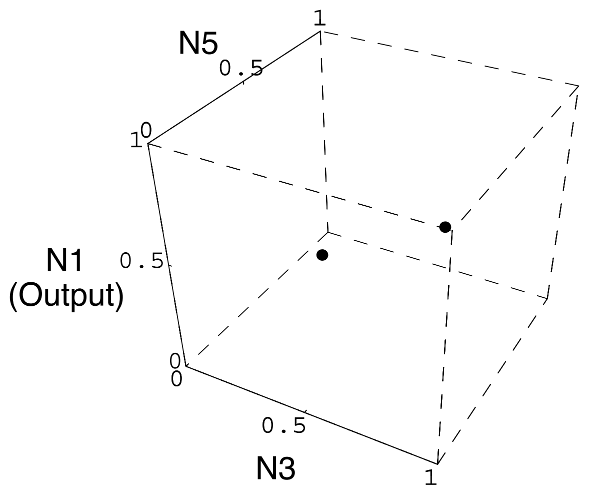
A


0* Positive
1* Positive
Trigger
N1
(Output)
Desired
Sequence
0 0 0 0 0 0 1 1 1 1 1 1 1
Reinforcement
N2
N3
N4
N5

B

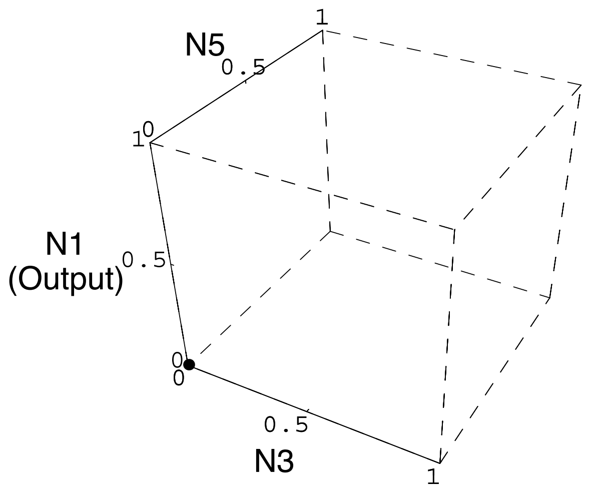

1* Positive
0* Positive
Trigger
N1
(Output)
Desired
Sequence
1 1 1 1 1 1 0 0 0 0 0 0 0
Reinforcement
N2
N3
N4
N5

A



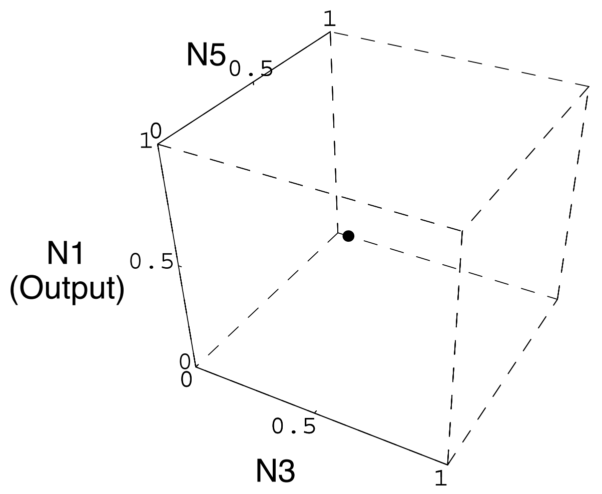
N5
0.5
1
0
1
N1
(Output)
0.5
0
0
0.5
N3
1

B



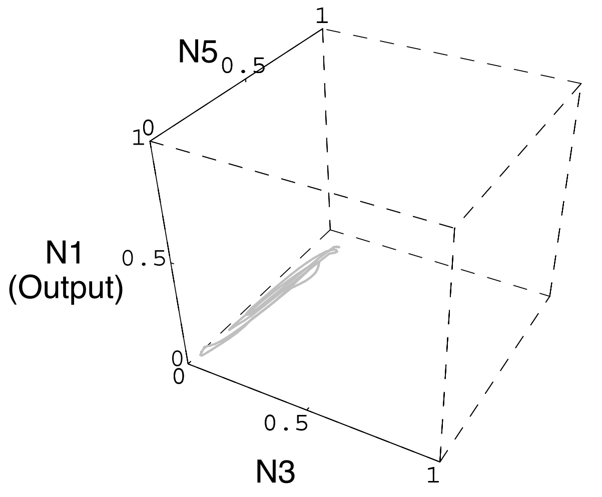
N5
0.5
1
0
1
N1
(Output)
0.5
0
0
0.5
N3
1

C



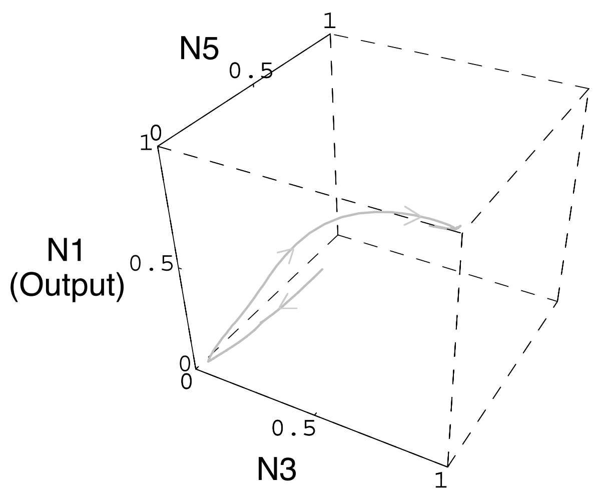
N5
0.5
1
0
1
N1
(Output)
0.5
0
0
0.5
N3
1

D

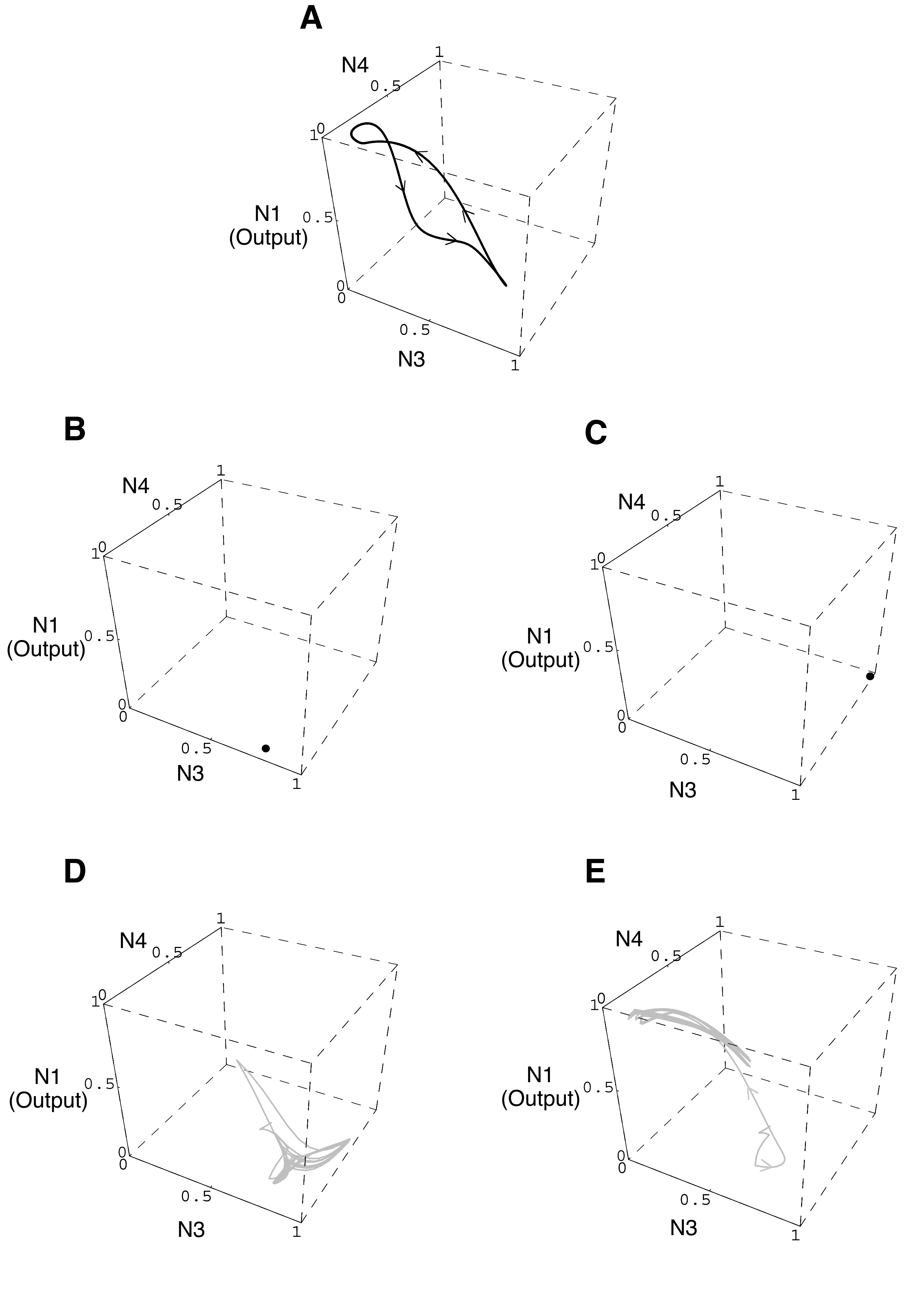


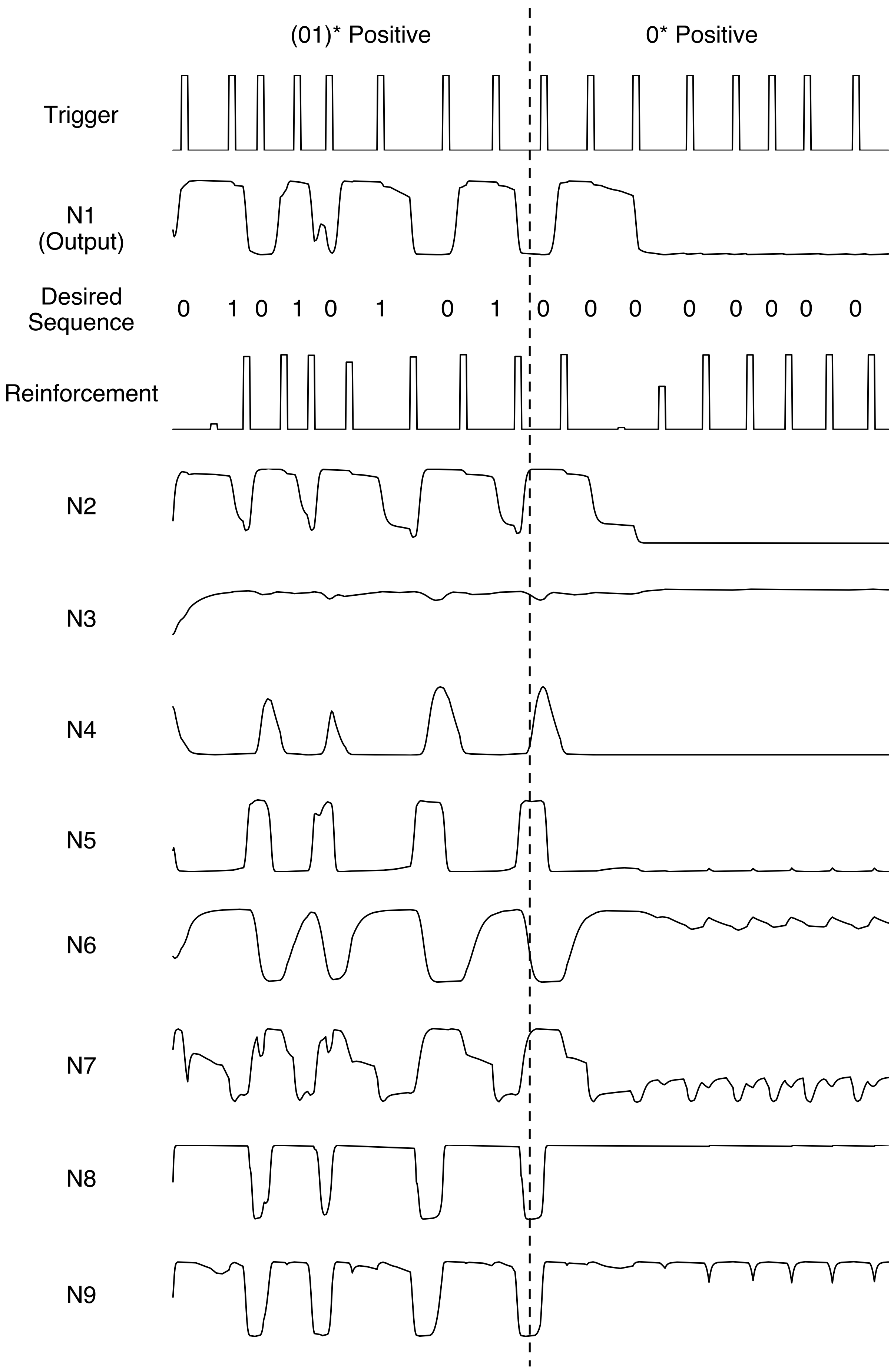
N5
0.5
1
0
1
N1
(Output)
0.5
0
0
0.5
N3
1

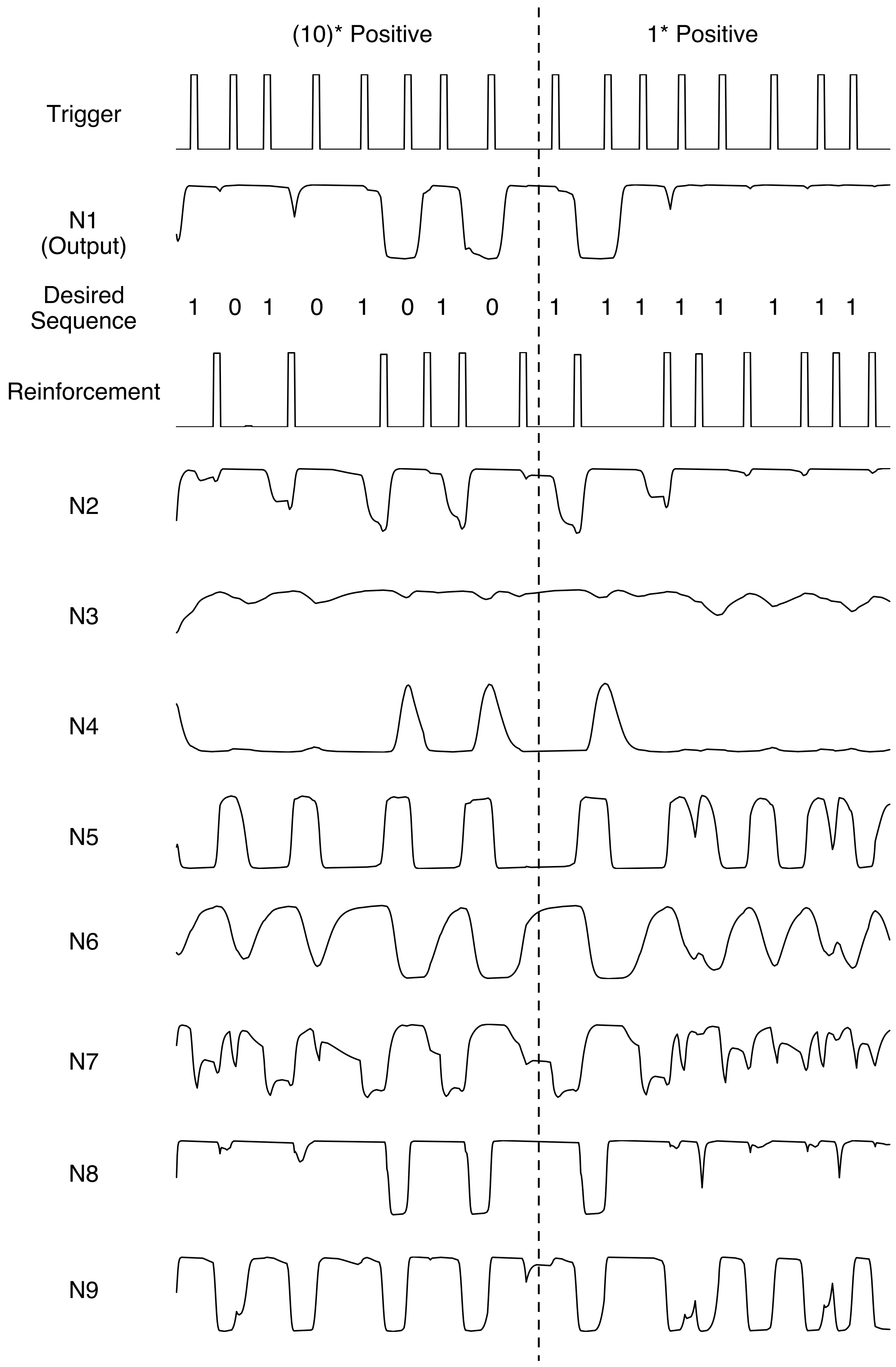
E

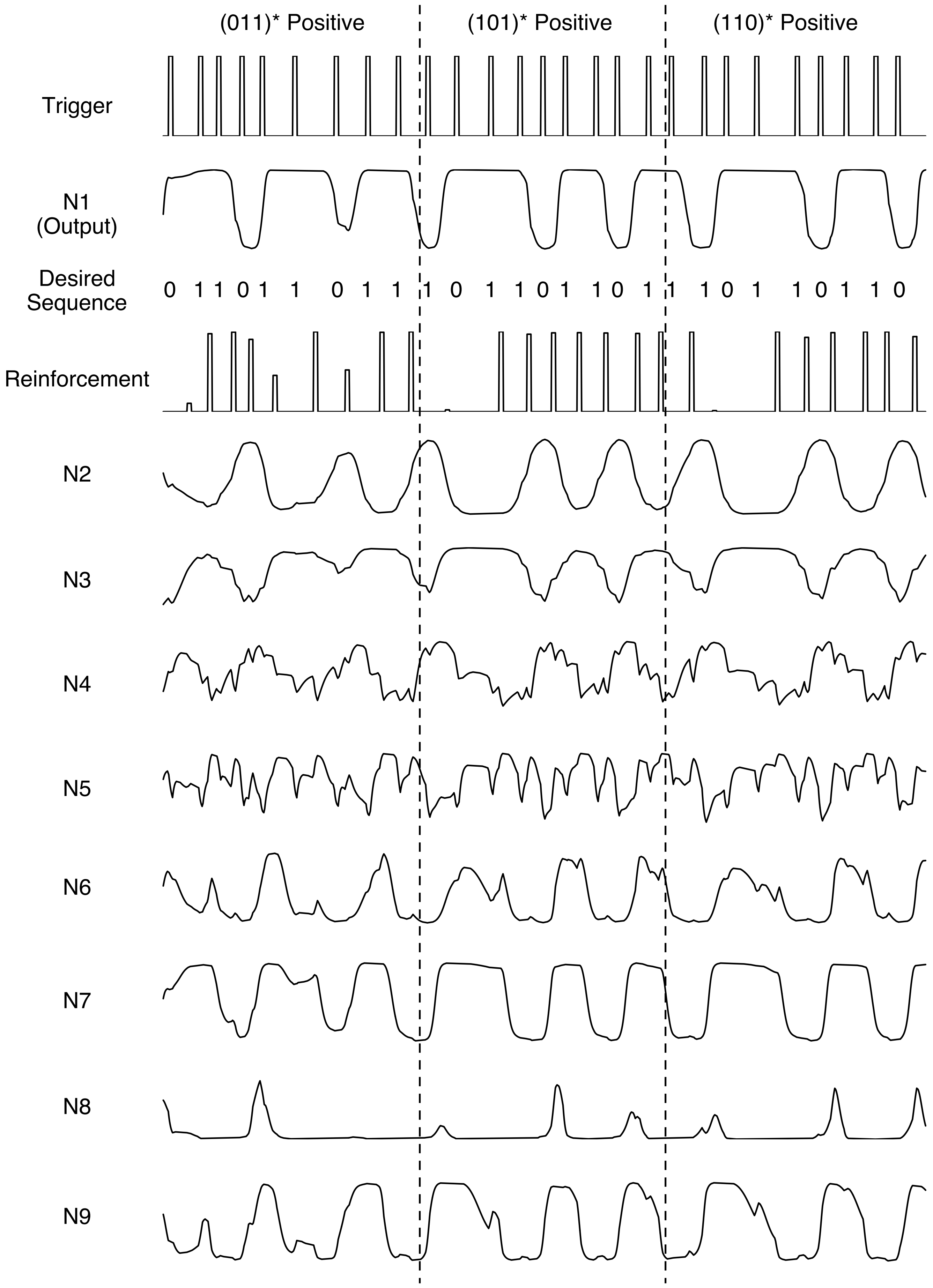


N5
0.5
1
0
1
N1
(Output)
0.5
0
0
0.5
N3
1


A
B
C
D
E
N4
0.5
1
0
N1
(Output)
N3


(01)* Positive
0* Positive
Trigger
N1
(Output)
Desired
Sequence
0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0
Reinforcement
N2
N3
N4
N5
N6
N7
N8
N9


(10)* Positive
1* Positive
Trigger
N1
(Output)
Desired
Sequence
1 0 1 0 1 0 1 0
1 1 1 1 1 1 1 1
Reinforcement
N2
N3
N4
N5
N6
N7
N8
N9


(011)* Positive
(101)* Positive
(110)* Positive
Trigger
N1
(Output)
Desired
Sequence
0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0
Reinforcement
N2
N3
N4
N5
N6
N7
N8
N9