DISSERTATION

# AN ECHO STATE MODEL OF NON-MARKOVIAN REINFORCEMENT LEARNING

Submitted by

Keith A. Bush

Department of Computer Science

In partial fulfillment of the requirements

for the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Spring 2008

COLORADO STATE UNIVERSITY

January 31, 2008

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY KEITH A. BUSH ENTITLED AN ECHO STATE MODEL OF NON-MARKOVIAN REINFORCEMENT LEARNING BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work

Adviser

Department Head

ABSTRACT OF DISSERTATION

# AN ECHO STATE MODEL OF NON-MARKOVIAN REINFORCEMENT LEARNING

There exists a growing need for intelligent, autonomous control strategies that operate in real-world domains. Theoretically the state-action space must exhibit the Markov property in order for reinforcement learning to be applicable. Empirical evidence, however, suggests that reinforcement learning also applies to domains where the state-action space is approximately Markovian, a requirement for the overwhelming majority of real-world domains. These domains, termed non-Markovian reinforcement learning domains, raise a unique set of practical challenges. The reconstruction dimension required to approximate a Markovian state-space is unknown *a priori* and can potentially be large. Further, spatial complexity of local function approximation of the reinforcement learning domain grows exponentially with the reconstruction dimension.

Parameterized dynamic systems alleviate both embedding length and state-space dimensionality concerns by reconstructing an approximate Markovian state-space via a compact, recurrent representation. Yet this representation extracts a cost; modeling reinforcement learning domains via adaptive, parameterized dynamic systems is characterized by instability, slow-convergence, and high computational or spatial training complexity.

The objectives of this research are to demonstrate a stable, convergent, accurate, and scalable model of non-Markovian reinforcement learning domains. These objectives are fulfilled via fixed point analysis of the dynamics underlying the reinforcement learning domain and the Echo State Network, a class of parameterized dynamic system [30].

Understanding models of non-Markovian reinforcement learning domains requires understanding the interactions between learning domains and their models. Fixed point analysis of the Mountain Car Problem reinforcement learning domain, for both local and nonlocal function approximations, suggests a close relationship between the locality of the approximation and the number and severity of bifurcations of the fixed point structure. This research suggests the likely cause of this relationship: reinforcement learning domains exist within a dynamic feature space in which trajectories are analogous to states. The fixed point structure maps dynamic space onto state-space. This explanation suggests two testable hypotheses. Reinforcement learning is sensitive to state-space locality because states cluster as trajectories in time rather than space. Second, models using trajectory-based features should exhibit good modeling performance and few changes in fixed point structure.

Analysis of performance of lookup table, feedforward neural network, and Echo State Network (ESN) on the Mountain Car Problem reinforcement learning domain confirm these hypotheses. The ESN is a large, sparse, randomly-generated, unadapted recurrent neural network, which adapts a linear projection of the target domain onto the hidden layer. ESN modeling results on reinforcement learning domains show it achieves performance comparable to lookup table and neural network architectures on the Mountain Car Problem with minimal changes to fixed point structure. Also, the ESN achieves lookup table caliber performance when modeling Acrobot, a four-dimensional control problem, but is less successful modeling the lower dimensional Modified Mountain Car Problem. These performance discrepancies are attributed to the ESN's excellent ability to represent complex short term dynamics, and its inability to consolidate long temporal dependencies into a static memory.

Without memory consolidation, reinforcement learning domains exhibiting attractors with multiple dynamic scales are unlikely to be well-modeled via ESN. To mediate this problem, a simple ESN memory consolidation method is presented and tested for stationary dynamic systems. These results indicate the potential to improve modeling performance in reinforcement learning domains via memory consolidation.

Keith A. Bush
Department of Computer Science
Colorado State University
Fort Collins, Colorado 80523
Spring 2008

ACKNOWLEDGEMENTS

I would like to express thanks to the following people who contributed in some way to the completion of this thesis:

Chuck, for taking me on an incredible graduate school journey;
Carol, for being a surrogate mother to me in Fort Collins;
Mark, Mike, Thomson, Andres, Nate, Monte, Crystal, and all the graduate students at CSU, for the countless questions, answers, and information sharing it takes to make the gears of research turn in academia;
my parents, for instilling in me a respect of education;
and I would particularly like to acknowledge Anna, for helping me through those final, crucial months of writing during a very difficult time in my life.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# Chapter 1

# Introduction

A driving force in reinforcement learning research is the growing need for intelligent, autonomous control strategies that operate in real-world domains. Many real-world problems involve a high-dimensional, nonlinear, dynamic, and continuous-valued state-space that is only partially observable through signals containing some degree of noise; a mathematically challenging place, indeed. Applying reinforcement learning within this domain, therefore, requires function approximation techniques that learn even when the domain's simplifying assumptions, the popular "gridworld", for example, have been relaxed and replaced by parameterized, discrete-time dynamic systems which are both nonlinear and non-Markovian.

These function approximation requirements are, unfortunately, difficult to fulfill due to mathematical and practical challenges of adapting parameterized dynamic systems, which are notoriously sensitive to changes in the observable dynamics. Overcoming these challenges and satisfying the unique requirements of non-Markovian reinforcement learning domains are the foundation of the research problem studied in this document. This introductory chapter is organized into sections presenting the research motivation, description of the problem, objectives, approach, and an overview of the remainder of this document.

## 1.1 Linking Histories with Horizons

In the physical world, time is the medium of change. Knowledge of temporal structure, therefore, is a core component of intelligence. Temporal knowledge emerges through decomposition of history, extraction of its core structural features, and from these features synthesis of accurate, robust predictive models. The task literally is, as Elman titled his seminal work of recurrent network modeling, *"Finding Structure in Time"* [18].

In a sense, the ability to accurately predict the future via a model is a measure of intelligence. A model is only as sophisticated as its mathematical foundations. The accuracy of a system's model is directly dependent on the extent to which that system is mathematically described and understood. The accuracy of future predictions derived from this model, therefore, are similarly dependent. Increasing model prediction accuracy is a means of increasing system understanding and, arguably, intelligence.

Temporal structure in the real-world, however, cannot simply be extracted from a static, albeit complex, historical record. Even the most precise predictive ability is wasted without the corresponding ability to act on the prediction. Goal-directed action, therefore, must supplement knowledge. Uniting these tools defines the challenge of intelligent decision making in the real-world.

Actions evoke changes in the world's structure. Goal-directed action is analogous to a premeditated change of the world's future where the change moves the world closer to an intended goal. Knowledge of temporal structure, therefore, requires understanding beyond that of extrapolating future trends based on structure embedded in history. It requires the profound ability to premeditate possible futures over a temporal horizon and to organize these futures based on their utility of achieving the goal.

While seemingly daunting, the problem of constructing models that link real-world histories with premeditated, unrealized horizons is mathematically well-defined. The process, termed *non-Markovian reinforcement learning* is the marriage of two independent research fields: nonlinear system identification, also termed *predictive modeling*, which studies the accurate prediction of future events based on mathematical models extracted from historical data, and *reinforcement learning*, which is a technique for assessing and organizing utility embedded in the premeditation of possible futures over a temporal horizon. For more than twenty years these fields, respectively, have been fertile ground for research. The need for intelligent, autonomous control strategies, however, is driving these long-distinct fields of research together.

The research proposed in this document investigates a small subset of challenges that are important, and in some cases unique, to the unification of predictive modeling and reinforcement learning. To develop these challenges, however, it is first necessary to outline the premise of reinforcement learning and non-Markovian state spaces, as well as the myriad of technical challenges that exist in modeling these spaces.

## 1.2 Non-Markovian Reinforcement Learning Domains

To understand the focus of this research, this document presents a sequence of concepts, each building on the previous to arrive at the research problem of interest. First I introduce canonical reinforcement learning, which exhibits a Markovian state-space. I then remove the assumption of a Markovian state-space and discuss what challenges this poses for reinforcement learning—the non-Markovian reinforcement learning problem. I also discuss the restrictions this imposes on practical non-Markovian reinforcement learning implementations—adaptive, parameterized dynamic systems.

The assessment and organization of goal-achieving value[1] embedded in premeditated futures can be achieved through the state-action-value model of learning. In this model, the value associated with a state varies according to the action taken by the agent existing in that state. In other words the value is a function of both the state *and* the action taken while in that state.

For this functional relationship to be correct the state-space must be *complete*. By completeness it is assumed that all information necessary to determine the next state of the world is contained in the current state. Completeness also assumes that a world can only exist in one state and that a unique action drives the current state into one and only one new state. In a complete state-space, therefore, the agent can influence the next state of the world only through its action selection. A complete state-space exhibits the Markov property [73], hence the term Markovian state-space. The Markov property is usually assumed in reinforcement learning and, therefore, is omitted when describing a problem

[1] In the reinforcement learning literature, utility is termed *value*. Canonical reinforcement learning nomenclature will be used throughout this document.

having a Markovian state-space.

Reinforcement learning provides the mathematical description of how to construct state-action-values. The description is based, as the name implies, on an agent receiving reinforcement signals in response to interaction with the environment. Reinforcement signals are transformed into state-action-values as follows. Each unique action transitions the agent from the current state into a new state. The action and new state, together, form a state-action pair. The reinforcement signal issued by the environment is a scalar function of the state-action pair as well as the desired goal, which is also a function of state-action pair.

The state-action-value is the maximum sum of future reinforcement signals that can be received when transitioning through the state-action space. That is, the current state-action-value is defined by summing over the reinforcement signals received by a sequence of state-action pairs, assuming that at each transition the action maximizing state-action-value is selected.

State-action pairs that achieve the goal receive the maximum reinforcement signal. Any state-action chain maximizing the sum of reinforcement signals must include the goal. Therefore, the action maximizing the state-action-value of a state determines the action necessary to achieve the goal from that state. If the reinforcement signal mapping function is constructed such that the maximum value is zero (i.e., the goal state-action pair) and all other state-action pairs receive a value less than zero, then the state-action chain maximizing the sum of reinforcement signals will not only include the goal, but it will also be the shortest chain possible, and, therefore, optimal in the sense of minimum actions necessary to achieve the goal. This is the premise of reinforcement learning.

## 1.3 Complications of non-Markovian Domains

Theoretically, the state-action space must exhibit the Markov property in order for reinforcement learning to be applicable [73]. Empirical evidence, however, suggests that reinforcement learning also applies to domains when the state-action space is only approximately Markovian [12, 49, 50, 51].

The absence of complete state adds a significant complication to the reinforcement learning problem. The nonlinear system identification community resolves this problem with a theoretical tool, Takens theorem [74]. Takens theorem states that the Markov property can be approximately reconstructed for a non-Markovian domain by temporally embedding multiple, sequential incomplete states into a single higher-dimensional state. This can be thought of as transfer of temporal information to spatial information. Takens theorem proposes that there exists, for some unknown temporal length, an incomplete state embedding which possesses the Markov property and approximates the original state-space.

While Takens theorem provides a theoretical solution to the problem of incomplete state representation, it raises additional practical challenges. For a given problem, the length of the embedding is unknown *a priori* and can potentially be large. The dimensionality of the state-space is an important practical limitation in reinforcement learning because the size of the approximation of the state-action values grows, potentially, exponentially with the dimension of the state-space. Any but the smallest embedding is, therefore, undesirable as an approximate state. These two points suggest that approximation of state-action-values using adaptable, parameterized dynamic systems is desirable when learning via reinforcement signal in an incomplete state-space.

Parameterized dynamic systems alleviate both embedding length and state-space dimensionality concerns by adapting a compact, dynamic representation that spans the observable state-space over a potentially infinite time-span. Yet this solution comes at a price. Modeling non-Markovian reinforcement learning domains via adaptive, parameterized dynamic systems is characterized by numerous technical challenges: instability, slow-convergence, and high cost of either computational or spatial complexity.

## 1.4 Objectives

The long-range objective of this research is development of reinforcement learning methods applicable to the real-world. This goal is, at present, unrealistic. As with many scientific experiments, identifying a suitable level of problem constraint is difficult. One way to view this challenge is to imagine problem constraints as layers of insulation that protect the elegant mathematical principles from the unforgiving complexity, dimension, and chaos of the real-world. Peeling back these layers broadens the range of applicable problem domain at the expense of greater experimental complexity and reduced ability to analyze and understand experimental results.

A realistic objective for this research is to investigate, identify, and understand techniques that successfully model constrained non-Markovian reinforcement learning domains. Ideally, these techniques should be **stable**, **convergent**, **accurate** and **scalable**. The amount of problem constraint is determined by the technique of interest. Constraints can be relaxed until the technique fails. The following challenges of applying reinforcement learning to real-world domains, therefore, evoke poignant, high-level questions that form the foundation of this research.

Dynamic side-effects: To what extent does instability, slow convergence, and cost of parameterized dynamic system adaptation influence reinforcement learning in non-Markovian domains? What are the consequences of these influences?
State-of-the-art: How has machine learning methodology addressed these modeling challenges? Are current methods sufficient for modeling non-Markovian reinforcement learning domains?
Next Generation: Can knowledge gained through the study of reinforcement learning within dynamic systems suggest improvements to existing methods?

In the following section the experiments and methods developed to achieve these objectives are described. Results of these experiments, their analysis, and the resulting contribution are also summarized.

## 1.5 Approach

The objectives of this research are to identify techniques that successfully model non-Markovian reinforcement learning domains. The approach used to fulfill these objectives relies on two tools—fixed point analysis of the dynamics underlying the reinforcement learning domain and the Echo State Network, a non-Markovian modeling architecture. Experiments and results using these techniques are developed below.

As a preliminary step to understanding non-Markovian domains, the interactions between Markovian reinforcement learning domains and their models were studied. During this step, it was assumed that reinforcement learning is a dynamic system, and it may be subjected to fixed point analysis. Fixed point analysis of the Mountain Car Problem reinforcement learning domain, when the domain is modeled via both local and non-local function approximations, provides useful generalizations of reinforcement learning dynamics. The locality of the function approximation determines both the number and impact of fixed point changes during learning; locality induces more, but less significant, changes to the fixed point structure; non-locality induces fewer, but more significant, changes to fixed point structure. Why is this occurring?

Research into this phenomena suggests a likely answer. Reinforcement learning domains exist within a fundamental feature space. In this space trajectories are analogous to states. The fixed point of the domain maps this dynamic space onto the state-space. Understanding this fundamental structure suggests two things: reinforcement learning is sensitive to state-space locality because states naturally cluster as trajectories in time more so than they group in space; domain models that utilize inherently trajectory based features should exhibit good modeling performance and very few changes in fixed point structure. With this insight, the remainder of the research focused on exploring the modeling performance of the Echo State Network (ESN) in multiple reinforcement learning domains: Mountain Car Problem, Modified Mountain Car Problem (i.e., single pendulum swing-up), and Acrobot (i.e., double pendulum swing-up).

The ESN is a large, sparse, randomly-generated, unadapted recurrent neural network, termed the *reservoir*. The ESN is trained by learning a set of linear weights, the *readout*, which project the target domain onto the reservoir, a dynamic basis, which generates trajectory based features, called *echoes*. The structure of the reservoir provides all of the properties desired for reinforcement learning: stability, scalability, and robustness.

Experimental results show that the ESN achieves performance comparable to lookup table and neural network architectures on the Mountain Car Problem and Acrobot but is less successful in modeling Modified Mountain Car. Further investigation attributes this failure to the ESN's inability to consolidate long temporal structure into a high-level, static memory. Without this type of memory, reinforcement learning domains exhibiting multiple dynamic scales are unlikely to be well-modeled via ESN.

An architecture for memory consolidation is proposed and tested in Chapter 7, termed *Mixture of Readouts* (MoR). MoR is an ESN analog to Mixture of Experts. The concept of MoR is that a stationary attractor may be decomposed into several smaller attractors. A fixed ESN is assumed to be able to achieve better modeling performance on each of these attractors individually, rather than the entire attractor when unique readouts are trained to model individual components of the attractor. Results gathered for both the Lorenz and Mackey-Glass attractors indicate that MoR improves modeling performance, in some cases substantially, for both open-loop and closed-loop time-series prediction. The open-loop results suggest that the MoR could be used as a simple memory consolidation technique when modeling reinforcement learning domains.

## 1.6 Overview

The remainder of this document details the specific mathematics, relevant prior work, approach, and experimental case studies that address the objectives and results highlighted above. Chapters 2 and 3 progress through past research leading up to the formation of these research questions, including mathematical details and historical perspective of the trends within reinforcement learning and predictive modeling. Chapter 3 also provides detailed motivation for the ESN architecture chosen to model the reinforcement learning domain. Chapter 4 presents the reinforcement learning domain as a dynamic system modeling problem and also presents results illustrating the experimental techniques and challenges faced in this research. Full non-Markovian case studies for the Mountain Car Problem are given in Chapter 5 and for the more complex Modified Mountain Car and Acrobot Problems in Chapter 6. Abstraction and memory consolidation techniques for non-Markovian predictive modeling architectures are suggested in Chapter 7. Research conclusions and an assessment of future research directions are presented Chapters 8 and 9, respectively.

# Chapter 2

# Reinforcement Learning Domains

Predictive modeling via adaptive, parameterized dynamic systems is the architectural glue that binds together an inherently non-Markovian world with a purely Markovian learning framework. The importance of predictive modeling to the non-Markovian reinforcement learning problem merits a thorough review. This review is given in Chapter 3. Successfully modeling non-Markovian reinforcement learning domains in practice, however, depends on subtle, mathematical details buried within the reinforcement learning framework, and particularly, the interaction of these mathematics with non-Markovian state-spaces. This chapter elucidates these mathematics, moving through the reinforcement learning framework, non-Markovian domains, nonlinear dynamic systems, and non-Markovian reinforcement learning domains.

## 2.1 Definition of the Problem

Choosing appropriate, goal-directed actions is the heart of intelligent, real-world interaction. Reinforcement learning is a technique for optimally solving multistep decision problems with mathematical foundations rooted in the theory of dynamic programming. Reinforcement learning is best characterized as learning through interaction with an environment [73].

The characterization has five components: *agent*, *environment*, *reward function*, *value function*, and *policy function*. The *agent* is the entity that observes the environment, learns, make decisions, and acts. The *environment* is the world, or a model of the world. The *reward function* is a mapping of the value of the environment's current state with respect to a desired goal, which is represented as a scalar reinforcement signal, also termed the reward signal. The *value function* is a mapping of the environment's current state onto the sum of expected reinforcement signals received along a trajectory that is initialized at the current state.

The goal of reinforcement learning is to learn a *policy function*, or simply *policy*, which is a mapping from the current state to an action, such that the expected sum of future reinforcement signals is maximized. This process directly implies learning from experience by interacting with the environment. The agent observes the state of the world, performs an action, and observes the reinforcement signal corresponding to the world's new state. If the agent selects actions that maximize the reinforcement signals then over time the value function will converge to the maximum expected sum of future reinforcement signals.

While reinforcement learning defines the means of characterizing the optimal solution of a multistep decision problem, it does not specifically outline how to build this solution. Reinforcement learning only outlines a method for approximating the *value function* when the optimal policy (i.e., the action that exactly maximizes the reward signal for the current state) is known. The optimal policy cannot be guaranteed without the exact *value function*. This argument introduces a paradox—how can one be known without the other?

### 2.1.1 The Actor-Critic Method

A solution to this paradox is termed the *actor-critic* method for reinforcement learning. The *policy*, or **actor**, is a mapping of the current state onto the action that maximizes the value function. The **critic** is simply another name for the *value function*. The input to the critic is dependent on the type of reinforcement learning technique used. One possibility is to use the current state, which is the notation in this section. Another possibility is to input both the current state and action, as in SARSA [73], outlined in the next section. In practice, for non-trivial sized state vectors, $\mathbf{s}$, mappings are approximated via parameterized functions. The policy is denoted $\pi(\mathbf{s})$. The value function is denoted $V(\mathbf{s})$. These functions are adapted iteratively.

If the adaptation is slow enough and if the parameterized functions are capable of representing actor and critic functions, respectively, then over many interactions with the environment, these functions will converge to their optimal representations, denoted, $\pi^*(\mathbf{s})$ and $V^*(\mathbf{s})$, respectively. The process is one of mutual information sharing. The critic builds a representation of expected future rewards associated with actions taken from the current state. The actor uses this representation to learn the action that maximizes future rewards. As the actor biases action selection, the critic biases the accuracy of its representation to those states visited most often, and therefore, to the rewards received in those states.

The actor-critic method provides a conceptually well-defined mechanism for converging to the optimal policy, $\pi^*(\mathbf{s})$. Mathematically, however, there exist a number of challenges to overcome. Many implementations of reinforcement learning exist that differ with respect to the structure of the value function and the manner in which the actor and critic functions are updated. In this thesis, state-action-reward-state-action (SARSA) method is used.

### 2.1.2 SARSA Reinforcement Learning

Reinforcement learning theory is predicated on the ability to construct the *policy* and *value function*. These mappings are explicit functions of states, $\mathbf{s}$. Reinforcement learning also assumes there exists an action-space where the resultant state, $\mathbf{s}_{t+1}$, is a mapping, $f$, from the current state, $\mathbf{s}_t$ and action, $a_t$,

$$\mathbf{s}_{t+1} \quad = \quad f(\mathbf{s}_t, a_t). \tag{2.1}$$

where only scalar actions, $a$, are considered. These assumptions require a reinforcement learning domain to be Markovian, which implies the state is completely observable and the state-transition function is known *a priori*.[1] Assuming a Markovian domain, the expected value of the sum of expected future rewards may be defined in a state-action-reward-state-action (SARSA) framework. The *value function*, $V^*$, becomes an *action-value function*, $Q^*$, which is a mapping of the state-action pair $(\mathbf{s}_t, a_t)$ onto the sum of expected future rewards received by selecting the action, $a_t$, based on the current policy, such that $a_t = \pi(\mathbf{s_t})$. The

[1]In this research only deterministic functions, $f$, are used for experimentation. In this case entries in the transition function take on values of either 1.0 or 0.0.

state-action-value function is defined by the recurrence:

$$Q^*(\mathbf{s}_t, a_t) \quad = \quad r_{t+1} + \gamma Q^*(\mathbf{s}_{t+1}, a_{t+1}) \tag{2.2}$$

where $r$ is the reinforcement signal and $\gamma$ is a parameter on the range $(0, 1)$ which guarantees a finite summation over an infinite horizon. Equation 2.2 assumes that $Q^*$ is known. Without *a priori* knowledge of $Q^*$, an approximation, $Q$, must be constructed iteratively. Assume that the current estimate of $Q$ contains error, $\delta_t$, at time $t$. Then Equation 2.2 can be rewritten as

$$\begin{aligned} Q(\mathbf{s}_t, a_t) + \delta_t &= r_{t+1} + \gamma Q^*(\mathbf{s}_{t+1}, a_{t+1}) \\ \delta_t &= r_{t+1} + \gamma Q^*(\mathbf{s}_{t+1}, a_{t+1}) - Q(\mathbf{s}_t, a_t) \\ \delta_t &= r_{t+1} + \gamma \left(Q(\mathbf{s}_{t+1}, a_{t+1}) + \delta_{t+1}\right) - Q(\mathbf{s}_t, a_t) \qquad (2.3) \end{aligned}$$

where $\delta_t$ is termed the *temporal difference error* or TD-error. Using TD-error to improve the approximation of state-action value is formulated as

$$Q(\mathbf{s}_t, a_t) = Q(\mathbf{s}_t, a_t) + \alpha\delta_t, \tag{2.4}$$

where $\alpha$ is a small constant value. Over many applications of TD-error to the current $Q$ estimate, the chosen action approaches the optimal action, defined as

$$a_t \quad = \quad \arg\max_a Q(\mathbf{s}_t, a). \tag{2.5}$$

It can be shown that $\delta_t \to 0$ as $t \to \infty$, and, therefore, $Q \to Q^*$ [73].

Equation 2.5 is modeled iteratively. It is assumed there exists some parameterized function describing the tendency, $p(\mathbf{s}_t, a_t)$, to select action $a_t$ given state $\mathbf{s}_t$. It is also assumed that the policy selects the action exhibiting the maximum tendency. To achieve the optimal policy, the tendency, $p$, is updated to increase or decrease the tendency of $a_t$ being selected when $\mathbf{s}_t$ is revisited at a future time. The update is,

$$p(\mathbf{s}_t, a_t) \quad = \quad p(\mathbf{s}_t, a_t) + \beta\delta_t, \tag{2.6}$$

where $\delta_t$ is the correction to the tendency for the current iterate and $\beta$ is a small constant value. Over many applications of TD-error, the tendency, $p$, of the action which maximizes

$Q$ for state $\mathbf{s}_t$ will converge to a maximum and the tendency of all other possible actions will tend toward a minimum. Thus, the policy $\pi$ will select the action maximizing the expected sum of future rewards.

Equations 2.2, 2.4, and 2.6 form the mathematical basis of SARSA reinforcement learning. Equations 2.4 and 2.6 provide the framework for implementing actor-critic reinforcement learning in terms of $\mathbf{s}_t$ and $a_t$. In practice, parameterized, generalized function approximations serve as functions $Q(\mathbf{s}_t, a_t)$ and $p(\mathbf{s}_t, a_t)$. Modification of $Q$ and $p$ based on TD-error (i.e., Equations 2.4 and 2.6) depend on the specific function approximation technique.

Most real-world problems are constrained such that the *agent* cannot observe the exact state of the world, $\mathbf{s}$. Often only an incomplete or partial state, $\tilde{\mathbf{s}}$, is observable. This type of domain is one type of partially observable Markov decision problem or non-Markovian domain. In this case, it is no longer possible to explicitly represent future rewards, which violates the most basic premise of reinforcement learning. Fortunately, this violation can be overcome by approximately reconstructing the underlying Markovian domain. Techniques for doing this are described in the next section.

## 2.2 Applied Non-Markovian Reinforcement Learning

The non-Markovian reinforcement learning domain is a hybrid of constructs fulfilling multiple tasks: approximate reconstruction of complete state-spaces from incomplete observations via temporal embeddings, automatic adjustment of temporal embedding lengths from observed dynamics, and formation of actor and critic mappings from the reconstructed state-space. This section steps through these constructs, culminating in the formal notations of the non-Markovian reinforcement learning problem.

### 2.2.1 Non-Markovian Domains

A Markovian system is characterized by a state vector $\mathbf{s}_t$ which explicitly contains all information such that there exists a mapping, $f$, of the next system state, $\mathbf{s}_{t+1} = f(\mathbf{s}_t)$. This is termed *complete state*. In contrast, non-Markovian domains are characterized by an observable state vector, $\tilde{\mathbf{s}}_t$, for which no mapping, $f$, exists that satisfies $\tilde{\mathbf{s}}_{t+1} = f(\tilde{\mathbf{s}}_t)$. This is often termed *incomplete state*.

As stated in the previous section, incomplete state violates a fundamental assumption in the derivation of SARSA, Equation 2.1. There does, however, exist a means of circumventing this violation. The complete system state can be approximately reconstructed from a trajectory of observable, incomplete states. This approach is motivated by Takens' Theorem [74].

Takens' Theorem states that under certain circumstances (i.e., a noiseless system observed with infinite precision), the state-space underlying a time-series can be reconstructed by projecting the time-series into higher dimension. The projection of the time-series into state-space is solved by introducing *delay coordinates*. The time-series' complete state, $\mathbf{s}_t$ can be reconstructed via $\tilde{\mathbf{s}}_t^E$ where $\tilde{\mathbf{s}}_t^E$ is an $E$-dimensional point defined as $\tilde{\mathbf{s}}_t^E = \{\tilde{\mathbf{s}}_t, \tilde{\mathbf{s}}_{t-1}, ..., \tilde{\mathbf{s}}_{t-(E-1)}\}$.

The minimum value $E$ that reconstructs the system's underlying state-space is termed the *intrinsic embedding dimension*, $E^*$. Takens' Theorem guarantees that an embedded state-space of dimension $E^*$ preserves the dynamics of the original time-series. A system's embedding dimension, however, is most often not known *a priori*. Empirically estimating the embedding dimension via correlation dimension is computationally expensive and the technique does not scale for high-dimensional embeddings [71].

Takens' Theorem implies, therefore, that a non-Markovian, incomplete state-space can be transformed, approximately, into a complete, Markovian state-space by projection of the observable, incomplete state-space into a higher dimensional embedding space. This theorem motivates the use of tap-delay input streams and recursive state descriptions for nonlinear system identification problems by supposing that these mechanisms act as an explicit or implicit embedding, respectively. Adaptive, parameterized dynamic systems utilizing recursive state descriptions are particularly useful in that some can, in theory, approximately reconstruct a complete state-space from incomplete observations while simultaneously tuning the implicit embedding dimension to fit the underlying dynamics. These traits will be discussed in greater detail in Chapter 3

### 2.2.2 Nonlinear Dynamic Systems

In previous sections the concept of state transition was introduced as a discrete mapping $f$ such that $\mathbf{s}_{t+1} = f(\mathbf{s}_t, a_t)$.[2]. Many real-world domains, however, are not discrete. Real-world domains are modeled by differential equations. Staying in line with notation already introduced, differential equations have the form:

$$\begin{aligned} \frac{\partial \mathbf{s}}{\partial t} &= \dot{\mathbf{s}}, \\ \dot{\mathbf{s}} &= f(\mathbf{s}). \end{aligned} \tag{2.7}$$

Generally, when $f$ is a nonlinear mapping, no analytical solution to Equation 2.7 exists. The system, however, can be discretized and simulated via numerical integration. One simple technique for doing this is the Euler method:

$$\begin{aligned} \frac{\mathbf{s}_{t+1} - \mathbf{s}_t}{\Delta t} &= f(\mathbf{s}_t), \\ \mathbf{s}_{t+1} &= \Delta t f(\mathbf{s}_t) + \mathbf{s}_t, \\ \mathbf{s}_{t+1} &= \mathbf{f}(\mathbf{s}_t), \end{aligned} \tag{2.8}$$

where $\Delta t$ is a discrete, typically small, unit of time. Equation 2.8 conforms to the notation introduced in previous sections to describe evolution of state.

There potentially exists enormous complexity within Equation 2.7. Imagine that the system state is comprised of $N$ dimensions. Then Equation 2.7 defines an $N$-dimensional vector field (i.e., the instantaneous direction and velocity of the evolution of the system). For non-trivial systems $f$ is a high-dimensional, parameterized nonlinear function. The structure of the vector field, $\dot{\mathbf{s}}$, provides insight into the potential long-term behavior of the system. Analysis of dynamic systems is a large, diverse field of research, but a few basic concepts are sufficient for providing context with respect to this research.

The study of vector fields centers around two topographic features. The first feature exists where $\dot{\mathbf{s}} = \mathbf{0}$. This is a *fixed point.* The second feature is the closed trajectory, commonly called a *limit cycle.* Both the fixed point and limit cycle indicate equilibria in

[2]In the general case, $a_t$ is assumed to an element of the state, $\mathbf{s}_t$.

the vector field where the long-term behavior of the system is well-defined, either constant (static equilibrium) or periodic (dynamic equilibrium).

As important as the location of equilibria is the qualitative shape of the vector field near these features. If the vector field points toward an equilibrium it is classified as an attractor (i.e., an attractive, or stable, equilibrium). Otherwise, if the vector field points away from the feature, it is classified as a repellor, or unstable equilibrium. Both fixed points and limit cycles are classified in this way. This is a brief, informal address of a very challenging, mathematically rigorous subject [71].

The difficult, often maddening, aspect of dealing with parameterized differential equations is that the equilibria of the vector field can change as the parameters change. This change may be the number, location, or stability of the equilibria, or some combination of changes. When the features of a vector field change via parameter modification either in number, stability, or both, then the parameter configuration evoking this change is called a *bifurcation*. Bifurcations can be parameter configurations in which small perturbations of the parameters of $\mathbf{f}$ (Equation 2.8) induce large qualitative changes in the evolution of the dynamic system (i.e., vector field, $\dot{\mathbf{s}}$).

The vector fields of dynamic systems, specifically their equilibria and bifurcations, are critical concepts to understanding the intention of this research and are developed further when non-Markovian modeling architectures are introduced in succeeding sections. These architectures are built upon parameterized differential equations. Training methods for these architectures constitute directed walks through the parameter space, potentially exposing these architectures to bifurcations. The impact of bifurcations on non-Markovian training methods is a fundamental exploration of this work.

### 2.2.3 Modeling Reinforcement Learning as a Dynamic System

Reinforcement learning domains are dynamic systems. What makes these domains challenging is that the recurrence describing the current prediction target, which determines the current decision, is dependent on future rather than historic state. Compare this with the general nonlinear dynamic system modeling task, the goal of which is to learn a recursive function that reproduces a known time-series with minimal error. In the case of reinforcement learning, however, the approximate recursive function actually generates the time-series as it learns.

The goal of learning is to reproduce the optimal time-series, which is defined by the optimal policy and value functions, respectively. Samples of this time-series and these functions are, unfortunately, unknown. Learning is shaped, rather, by the reinforcement signals. The trajectory of the approximate recursive function is driven toward the optimal trajectory via reinforcement signals. During learning, the underlying attractor of the dynamic system described by the approximate recursive function changes. Reinforcement learning problems, therefore, inherently exhibit nonstationary attractors.

To better comprehend the dynamics of reinforcement learning, it is necessary to derive and fully expand an approximation to the optimal state-action value function, $Q^*$, Equation 2.9. The value of the target, $Q^*$, is a sum of expected future rewards over a potentially infinite horizon. If we assume a finite horizon of $k$ future rewards, the result is the approximate state-action-value function, $Q$, shown in Equation 2.10.

$$\begin{aligned}
Q^*(\mathbf{s}_t, a_t) &= r_{t+1} + \gamma Q^*(\mathbf{s}_{t+1}, a_{t+1}) \qquad (2.9)\\
&= r_{t+1} + \gamma(r_{t+2} + \gamma Q^*(\mathbf{s}_{t+2}, a_{t+2}))\\
&= r_{t+1} + \gamma r_{t+2} + \gamma(\gamma(r_{t+3} + \gamma Q^*(\mathbf{s}_{t+3}, a_{t+3}))\\
&\vdots\\
&= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{k-1} r_{t+k} + \gamma^k Q^*(\mathbf{s}_{t+k}, a_{t+k})\\
&\approx r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{k-1} r_{t+k} \qquad (2.10)
\end{aligned}$$

Assuming a Markovian state-space, a function, $f_1$, may be constructed that maps the current state, $\mathbf{s}_t$, onto the future state, $\mathbf{s}_{t+1}$, as in Equation 2.11,

$$\mathbf{s}_{t+1} = f_1(a_t, \mathbf{s}_t). \qquad . \quad (2.11)$$

Using Equation 2.11, the future reward, $r_{t+1}$, can be defined as a function, $f_2$, of known quantities—the current state and action,

$$\begin{aligned} r_{t+1} &= f_2(a_t, \mathbf{s}_{t+1}) \\ &= f_2(a_t, f_1(a_t, \mathbf{s}_t)). \qquad (2.12) \end{aligned}$$

Substitution of Equations 2.11 and 2.12 into Equation 2.10 yields the form of a parameterized dynamic approximation to the reinforcement learning problem,

$$\begin{aligned} Q(\mathbf{s}_t, a_t) &= f_2(a_t, f_1(a_t, \mathbf{s}_t)) + \gamma f_2(a_{t+1}, f_1(a_{t+1}, \mathbf{s}_{t+1})) + \gamma^2 f_2(a_{t+2}, f_1(a_{t+2}, \mathbf{s}_{t+2})) \\ &\quad + ... + \gamma^{k-1} f_2(a_{t+k}, f_1(a_{t+k}, \mathbf{s}_{t+k})). \end{aligned}$$

The current action can be approximated by a function, $f_3$, of the current state, analogous to the actor, given by Equation 2.14,

$$\begin{aligned} a_t &= \arg\max_a (Q^*(a, \mathbf{s}_t)) \qquad (2.13) \\ &\approx f_3(\mathbf{s}_t). \qquad (2.14) \end{aligned}$$

Through substitution and expansion of functional arguments, an approximation of the entire reinforcement learning problem can be defined in terms of the current state, $\mathbf{s}_t$. This representation removes all dynamics, yielding a single pattern matching problem, presented in Equations 2.15—2.18.

$$\begin{aligned} Q(\mathbf{s}_t) &= f_2(f_3(\mathbf{s}_t), f_1(f_3(\mathbf{s}_t), \mathbf{s}_t)) + \gamma f_2(f_3(\mathbf{s}_{t+1}), f_1(f_3(\mathbf{s}_{t+1}), \mathbf{s}_{t+1})) \\ &\quad + \gamma^2 f_2(f_3(\mathbf{s}_{t+2}), f_1(f_3(\mathbf{s}_{t+2}), \mathbf{s}_{t+2})) \\ &\quad + ... + \gamma^{k-1} f_2(f_3(\mathbf{s}_{t+k}), f_1(f_3(\mathbf{s}_{t+k}), \mathbf{s}_{t+k})) \qquad (2.15) \\ \mathbf{s}_{t+1} &\approx f_1(f_3(\mathbf{s}_t), \mathbf{s}_t) \qquad (2.16) \\ \mathbf{s}_{t+2} &\approx f_1(f_3(f_1(f_3(\mathbf{s}_t), \mathbf{s}_t)), f_1(f_3(\mathbf{s}_t), \mathbf{s}_t)) \qquad (2.17) \\ &\vdots \\ \mathbf{s}_{t+k} &\approx f_1(f_3(f_1(f_3(f_1(f_3(...f_1(f_3(\mathbf{s}_t), \mathbf{s}_t)...)))))), f_1(f_3(f_1(f_3(...f_1(f_3(\mathbf{s}_t), \mathbf{s}_t)...)))) \\ &\qquad (2.18) \end{aligned}$$

Equations 2.15—2.18 provide a static view of the dynamics underlying reinforcement learning. Without knowledge of the actual function themselves, specific conclusions about the surface described by $Q(\mathbf{s}_t)$ are not possible. However, understanding how $Q$ depends on the functions defining state transitions, rewards, and action selection does imply certain difficulties of modeling reinforcement learning via function approximation.

The general architecture of Equation 2.15 is a sum of $k$ terms defining individual future rewards, analogous to Equation 2.10. Each of the terms is a function of the state at time $t+k$ in the future. Another gross feature of importance is the relationship between successive system states, $\mathbf{s}_{t+i}$, $i = 1, ..., k$. Each state further into the future is a function of the composition, $f_1(f_3(\cdot))$, taking the previous state as input. The final relevant feature of Equation 2.15 is that, as in Equation 2.10, each term is preceded by a constant, $\gamma$, taken to an exponent, $i-1$, $i = 1, ..., k$. As stated earlier, $\gamma$ is a scalar defined on the range $[0, 1]$. Therefore, each succeeding term of Equation 2.15 influences the target, $Q(\mathbf{s}_t)$, exponentially less than the previous term.

Examination of the dependence of Equation 2.15 is relevant with respect to several known sources: the system state $\mathbf{s}_t$ and functions, $f_1$, $f_2$, and $f_3$. For the purposes of this discussion, the problem can be simplified by assuming that iteration over the actions $a$ is feasible. This will determine the action maximizing $Q$ and function $f_3$ becomes exact.

Surprisingly, $Q(\mathbf{s}_t)$ is linear with respect to the reward function, $f_2$. Due to the recursive embedding of compositions, $f_1(f_3(\cdot), \mathbf{s}_t)$, $Q$ is exponentially dependent in time with respect to the state transition function, $f_1$. The state, $\mathbf{s}_t$, is exponentially dependent on past state through recursive application of the state transition function. In terms of potential dependence on approximated functional components, the surface of $Q(\mathbf{s}_t)$ is influenced by the topology of the reward function surface, and, if this shape is sensitive to input variation, the measurement accuracy of the system's state and approximation accuracy of the system's state transition function.

Expanding the recursion, as in Equations 2.15–2.18, also allows for a snapshot view of the bifurcation problem, particularly challenging when functions $f_1$ and $f_3$ are nonlinear. Determining analytically what impacts adaptation of functional parameters has on the prediction of $Q$ is very challenging because these adaptations have exponentially growing impact on the predictions of future rewards.

The impact of state and functional approximation error is particularly troublesome when attempting to guarantee stability. Guaranteeing system stability during adaptation of control decisions is a well-known problem that combines reinforcement learning with robust control theory. This problem has been studied previously [43] and will not be addressed here.

The implications of functional approximation error on the accuracy of $Q$ value predictions is significant, but not, actually, insurmountable in practice. There is no requirement that the approximation of $Q(\mathbf{s}_t)$ be constructed from recursions of function $f_1$, only that the information to be modeled takes this form. Further, the complexity of the surfaces of Equations 2.15—2.18 cannot be determined from the compositions of nonlinear functions. In practice, approximations of $Q$ may be much less sensitive to parameter adaptation and approximation error of $\mathbf{s}_t$. As evidence, given slow adaptation and multiple trials, function approximations have successfully modeled $Q$ for a range of control problems [40].

In a non-Markovian state-space, however, the dynamics governing the reinforcement learning domain are more complex. The premise of this dissertation focuses on the reinforcement learning consequences of substituting complete state $\mathbf{s}_t$ with a non-Markovian function approximation.

### 2.2.4 Non-Markovian State Approximation

When unavailable, complete state may be approximately reconstructed from a temporal embedding of prior observables,

$$\mathbf{s}_t \approx g_1(\tilde{\mathbf{s}}_t^E). \tag{2.19}$$

However, as discussed in a previous section, the intrinsic embedding dimension $E^*$ is almost always unknown *a priori*. Therefore, an implicit reconstruction of the state-space must be used,

$$\begin{aligned} \mathbf{s}_t &\approx g_2(\tilde{\mathbf{s}}_t, \mathbf{x}_t) \\ \mathbf{x}_t &= g_3(\tilde{\mathbf{s}}_{t-1}, \mathbf{x}_{t-1}), \end{aligned} \tag{2.20}$$

where $\tilde{\mathbf{s}}_t$ is the observable, incomplete state and $\mathbf{x}_t$ is a recursive, contextual representation of past observables. In this format the parameterized dynamic system in Equation 2.20 represents all historic information necessary to reconstruct the transition function of the system state, $\mathbf{s}_t$.

Combining Equation 2.20 with Equation 2.11 yields,

$$\begin{aligned} \mathbf{s}_{t+1} &\approx f_1(a_t, g_2(\tilde{\mathbf{s}}_t, \mathbf{x}_t)) \\ \mathbf{x}_t &= g_3(\tilde{\mathbf{s}}_{t-1}, \mathbf{x}_{t-1}). \end{aligned}$$

Assume that the parameterized dynamic system is adapted in such a way that it encapsulates the necessary information of the intrinsic embedding dimension, $E^*$, giving,

$$\mathbf{s}_{t+1} \approx f_1(a_t, g_2(\tilde{\mathbf{s}}_t, g_3(\tilde{\mathbf{s}}_{t-1}, g_3(\tilde{\mathbf{s}}_{t-2}, g_3(\mathbf{s}_{t-3}, g_3(..., g_3(\mathbf{s}_{t-E^*}, \mathbf{x}_{t-E^*})...)))).$$

Substituting for action, $a_t$,

$$\begin{aligned} \mathbf{s}_{t+1} \approx\ & f_1(f_3(g_2(\tilde{\mathbf{s}}_t, g_3(\tilde{\mathbf{s}}_{t-1}, g_3(\tilde{\mathbf{s}}_{t-2}, g_3(\tilde{\mathbf{s}}_{t-3}, g_3(..., g_3(\tilde{\mathbf{s}}_{t-E^*}, \mathbf{x}_{t-E^*})...))))), \\ & g_2(\tilde{\mathbf{s}}_t, g_3(\tilde{\mathbf{s}}_{t-1}, g_3(\tilde{\mathbf{s}}_{t-2}, g_3(\tilde{\mathbf{s}}_{t-3}, g_3(..., g_3(\tilde{\mathbf{s}}_{t-E^*}, \mathbf{x}_{t-E^*})...)))). \end{aligned} \tag{2.21}$$

Comparing this result to Equation 2.16, it is humbling to consider the approximation error and sensitivity placed on function $g_3$, the parameterized dynamic system forming the implicit embedding of historical context. As with any function approximation, there are consequences of non-Markovian state approximation. First, approximation error will exist in the estimation of state, $\mathbf{s}_t$. The second consequence is substantially greater. Change in approximation of $\mathbf{s}_t$ grows exponentially with $E$ for function $g_3$. Drawing on the $Q$ approximation influence of $\mathbf{s}_t$, adaptation of function $g_3$ becomes problematic. These derivations imply important consequences; an adaptive parameterized dynamic system reconstructing complete state for approximation of $Q$ must: **1) be very accurate** and **2) undergo minimal adaptation, or ideally, no adaptation at all**.

The consequences of utilizing parameterized dynamic systems to model non-Markovian domains require a search for architectures that satisfy these two criteria. In Chapter 3, predictive modeling theory and architectures are reviewed with particularly emphasis placed on parameterized dynamic systems, concluding with a review of the Echo State Network, an architecture well-suited to satisfying the requirements stated above. In Chapters 5 and 6, the Echo State Network is studied as a model of several classic reinforcement learning problems in the non-Markovian domain.

# Chapter 3

# Predictive Modeling

The research proposed in this document is comprised of two techniques: non-Markovian reinforcement learning, described in the previous chapter, and the technique that bridges the gap between a non-Markovian environment and a Markovian learning representation, predictive modeling. Predictive modeling has a mature literature but most of this literature is irrelevant to the questions addressed in this work. However, to understand the relationship between predictive modeling and non-Markovian reinforcement learning, described in Chapter 2, it is important to review the literature as it developed through time. Predictive modeling is a classic example of iterative advancement—each breakthrough solved an existing problem only to introduce a more challenging problem.

The remainder of this chapter will first build a working vocabulary of predictive modeling. Specifically, I will develop taxonomies of architecture, training, and performance evaluation, as well as a brief discourse on the theoretical limits of predictability. An overview of predictive modeling advancements will precede an in-depth review of the literature involved with the most recent and powerful modeling methods, the recurrent and reservoir architectures.

## 3.1 Prediction Architecture Taxonomy

Predictive models are mathematical tools that forecast a system's approximate future state trajectory from its historical state trajectory. This technique is separate from, but often applied in parallel with, dynamic models based on first principles. In this scenario the predictive models are termed observers [?], however, the terminology is often interchangeable. Within the context of this research a predictive model extracts a system's description directly from the historical state with zero *a priori* knowledge of first principles. This definition includes complex, abstract systems where no first principle descriptions exist.

Predictive models have grown from the earliest linear regression models such as autoregressive moving average (ARMA) models to more recent and substantially more powerful nonlinear recurrent models. The progression from mathematically well-defined regression techniques to poorly understood nonlinear techniques does not come without cost. Rather, this transformation has driven predictive modelers to embrace the necessary, albeit nebulous, tools of the nonlinear optimization trade: heuristics and search. This difference in representation, linear versus nonlinear, and the unique mathematical tools accompanying each, demarks the most significant theoretical divide among predictive model architectures.

At the implementation level, however, predictive model architectures are also discriminated by their mechanism of feature extraction. Features are extracted from historical data in two forms—explicit and implicit. Explicit feature extraction builds temporal features by direct transformation of a relevant, finite-length sequence of historical data at each timestep. In contrast, implicit feature extraction builds an internal, dynamic representation of historical data which makes no rigorous assumption of relevant time frame.

Consider the space of predictive architectures. The primary parameters of this space are *model complexity*, linear or nonlinear, and *feature representation*, explicit or implicit. These discriminants are not mutually exclusive; nonlinearity and implicit features subsume linearity and explicit features, respectively. To make each architecture unique in this space, however, requires a third parameter, training method. Figure 3.1 depicts the space of predictive architectures. For brevity, Figure 3.1 only expands the training dimension for the major nonlinear recurrent neural network architectures.

The training dimension is multimodal and is best represented as a tree of algorithms. Each training mode implies a branch of this tree. In the taxonomy presented in Figure 3.1 the first mode of training is *model dynamics*—adaptive or stochastic. The second mode is the method of *cost function minimization*, which is often improperly referred to as the training mechanism. The leaves of this hierarchy are labeled with the predictive architec-

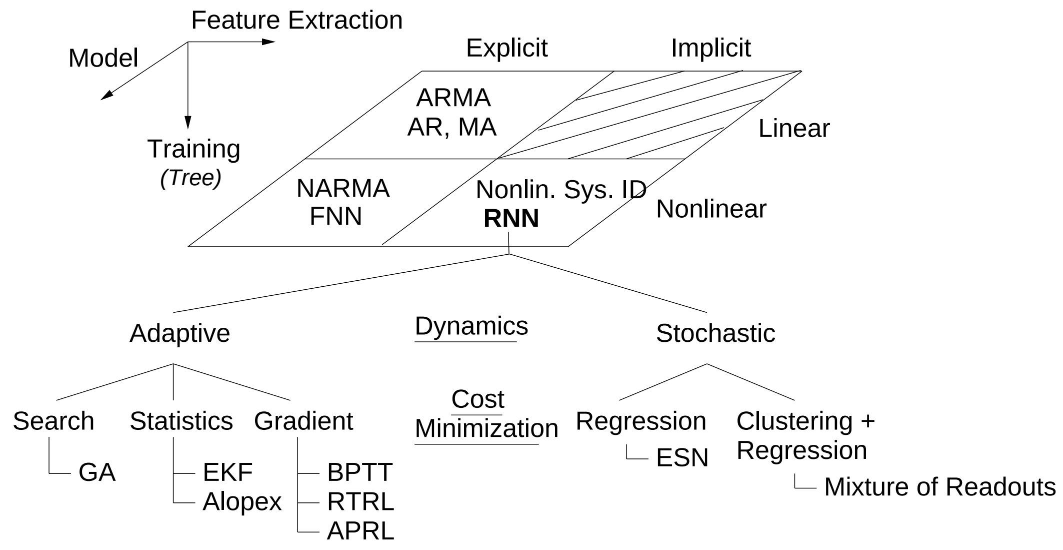

Figure 3.1: Representation of the taxonomy of predictive modeling. The space of predictive architectures is 3-dimensional, parameterized by model complexity, feature extraction technique, and training method. The training method dimension is multimodal and is expanded for the RNN architecture.

ture.

This decomposition is useful in this research in that it makes clear distinction between embedded, recurrent, and (stochastically-recurrent) reservoir architectures. These are the categories chosen to present relevant prior work. Using this spatial representation of predictive architectures, relevant prior predictive modeling research will be addressed in temporal progression of advancements. Relevant relationships between successive advances will be drawn between the three architectural features.

## 3.2 Prediction Performance Criteria

The literature discusses three performance attributes of predictive models: accuracy, trainability, and scalability. Respectively, these attributes attempt to quantitatively or qualitatively assess how accurately an architecture can predict future states, how reliably an architecture can reproduce predictions, and the dimension to which the state-space of a problem may grow before the architecture can no longer generate useful predictions. Also, it is important to note that there exist limits to the predictability of certain domains. The end of this section will briefly address literature relating the theoretical limits of predictability as a property of chaotic systems.

### 3.2.1 Accuracy

The broadest measurement of predictive model accuracy is *generalized mean-squared prediction error*. This measurement is defined as the mean-squared prediction error over a test dataset that is temporally contiguous with, but not included in, datasets used to train and validate the model. Define the model output at time $t$ as $\tilde{\mathbf{s}}(t)$ which approximates the system's true state, $\mathbf{s}(t)$. Then prediction error is

$$\mathbf{e}(t) \quad = \quad \mathbf{s}(t) - \tilde{\mathbf{s}}(t) + \eta(t), \tag{3.1}$$

where $\eta(t)$ is noise. This defines a single prediction error point of the network. Typically, recurrent networks model the target system's entire temporal trajectory, $\mathbf{s}(t)$, $t \in T_{test}$, such that expanding the cost of prediction error over an entire trajectory and computing the mean of the sum of squares of this trajectory gives the error measure desired:

$$E_{test}^{mse} \quad = \quad \frac{1}{T_{test}} \sum_{t=1}^{T_{test}} \mathbf{e}(t)^2. \tag{3.2}$$

Predictions are categorized in two very different ways, *open* versus *closed* loop, which are differentiated as follows. Model output is a function of the input $\mathbf{u}(t) = (\mathbf{s}(t), \mathbf{a}(t))$, where $\mathbf{a}(t)$ is external influence acting on the system, such that $\tilde{\mathbf{s}}(t) = f(\mathbf{u}(t))$. Open-loop predictions are computed by supplying the true state of the system at time $t$ as input for predictions at time $t+1$, giving $\mathbf{u}(t) = (\mathbf{s}(t), \mathbf{a}(t))$.

Closed-loop prediction error is computed by supplying the value of the model prediction at time $t-1$ as input for predictions at time $t$, therefore, $\mathbf{u}(t) = (\tilde{\mathbf{s}}(t-1), \mathbf{a}(t))$. In the general case, closed-loop error is significantly more representative of the model's predictive capability but is also more difficult to minimize and can lead to performance measurements that are wildly varying or numerically undefined. To avoid numerical instability problems that arise in closed-loop error measurement, researchers often substitute *closed-loop prediction steps*, $T_{test}^{closed}$ as the performance measure. $T_{test}^{closed}$ is an integer defining the number of closed-loop predictions made by the model before squared prediction error exceeded some threshold, $\epsilon$, which is formally defined,

$$T_{test}^{closed} = \arg\max_t \left(\mathbf{e}(t)^2 < \epsilon\right). \tag{3.3}$$

### 3.2.2 Trainability

Trainability is often measured in conjunction with accuracy, but it is a unique performance attribute. Trainability is a measure of the convergence speed and robustness of a training method. This measure may be generated numerous ways. For a given time-series, trainability could be the number of steps required to achieve minimum prediction error coupled with the percentage of random training instantiations that fall within $\epsilon$ of the minimum prediction error. A number of other metrics may satisfy this attribute. For instance, it is relevant to examine the cumulative distribution of minimum prediction errors achieved through random restart to highlight weaknesses in the architecture.

### 3.2.3 Scalability

Scalability is the ability of an architecture to maintain accuracy and trainability as problem size increases. Problem size is defined as the dimensionality of the problem's state-space. Like trainability, no clean definition of this metric exists. Alternatively, maintenance of accuracy is not necessary for an architecture to scale (i.e., exhibit scalability). For an architecture to scale, it need only exhibit predictive utility as problem size increases. Therefore, accuracy may decay as long as the architecture remains useful in a problem dependent context. If so, the architecture is scalable.

### 3.2.4 Limits of Predictability

While the theory of how it may be possible to learn a predictive model based on historical data is well-defined, there also exists a body of work dictating the theoretical limits of predictability. These limits apply to a specific class of dynamic systems—chaotic systems. While there exists disagreement of what exactly constitutes a chaotic system, three properties are almost universally accepted [71]. To qualify as chaotic, a system must be deterministic, exhibit aperiodic behavior in the long-term, and be sensitive to initial conditions. The third property is that which makes prediction of chaotic systems most troublesome. This sensitivity emerges from exponential divergence of trajectories initiated from neighboring states. This property is quantified by a positive *Liapunov exponent*.

Using the notation of Strogatz [71], given two initial system conditions, point $x_0$ and a neighboring point $x_0 + \delta_0$ such that perturbation $\delta_0$ is very small, then the distance separating these trajectories at the $n$-th iteration, $\delta_n$ is given by $|\delta_n| \approx |\delta_0| e^{n\lambda}$. The divergence at the $n$-th iterate is exponentially proportional to the initial perturbation. Parameter $\lambda$ is the Liapunov exponent of the system. A chaotic system must have a positive Liapunov exponent.

Positive Liapunov exponents are a serious limitation of predictive models. The consequences of this limitation with respect to this research, however, are not serious. The predictive model is used in this research as a mechanism of assembling historical information to predict a single time step in the future. Specifically, the model need only predict the 1-st iterate of the action-value function. It is the action-value, $Q$, itself which encapsulates system state information over the temporal horizon—not the predictive model. The effect of exponential divergence is minimized because the current state is observed at each iteration—closed loop iterative predictions are unnecessary. Therefore, the limitations of predictability imposed by Liapunov do not directly apply.

## 3.3 Embedded Architectures

Revisiting the space of predictive architectures shown in Figure 3.1, it is, as presented above, possible to describe an architecture by any of the three parameters: model, feature, or training method. However, discourse is most natural if the architectures are distinguished according to their feature space first, followed by differences in their model complexity and training. In this section, I will review the broadest classes of embedded architectures. A similar methodology applies to subsequent divisions of recurrent architectures.

### 3.3.1 Linear Models

The most common predictive architecture, and certainly the oldest, is the class of embedded linear architectures, used widely in control, economic forecasting, and scientific modeling, termed the auto-regressive moving-average (ARMA) model [46]. ARMA is a linear combination of two unique model architectures, the auto-regressive (AR) and moving-average (MA) models. These architectures are very simple. AR assumes that future state is a linearly weighted sum of a finite history. The AR model is uniquely determined by choosing parameter, $p$, defining the length of the finite history used for the prediction. Regression maps the historical patterns onto the future states over a sequence of training data, and this fit is optimal in the least-squared sense. The moving average architecture is similar, but defines future model state as a linearly weighted sum of $q$ historical means, where each mean, $\mu_i$, $i = 1, 2, ..., q$, is built by averaging $i$ historical state values starting at the current state. This architecture introduces a notation of frequency domain—low frequency information is captured by averages computed from long sequences and high frequency information is captured by averages computed from short sequences. Again, regression is used to fit the linear mapping. The ARMA model is a linearly combination of the AR and MA models.

Constructing and training an ARMA is inexpensive. The only training uncertainty is proper choice of values $p$ and $q$ such that generalization error is minimized. However, due to the model's training efficiency, iterative search of these parameters is often feasible, making a recurrent linear model of little use. This fact is the reason that the linear-recurrent space of models is neglected in Figure 3.1. Finding the optimal embedded linear model is tractable.

The primary drawback of the ARMA model is that it is linear. Real-world problems of interest are generally not linear, thereby limiting this model's utility. ARMA modeling can be found in some of the earliest literature, and has long been the model of choice in virtually every scientific field from engineering to economics. The limitations of the linear model, however, are a force of innovation in machine learning. The world is nonlinear, and it must be approximated by models of nonlinear complexity. The next section describes a successful architecture born by the limitations of the ARMA model. Many other linear model architectures are available, particularly those developed by the system identification field (i.e., extended Kalman filtering). For a complete treatment, see [46, 32].

### 3.3.2 Nonlinear Models

Training efficiency drops drastically when linear model complexity does not apply. The parameters of nonlinear function approximations, in general, cannot be found analytically. Among nonlinear function approximations, the artificial feed-forward neural network (FNN) trained via error-backpropagation [65] has gained wide popularity. The FNN is a parameterized general function approximation constructed as a multistep nonlinear transformation:

1. linear projection of $M$-dimensional inputs into $N$-dimensional space, $\mathbb{R}^M \rightarrow \mathbb{R}^N$
2. point-wise nonlinear transformation in high-dimension, $\mathbb{R}^N \rightarrow \mathbb{R}^N$
3. linear projection to $P$-dimensional output space, $\mathbb{R}^N \times \mathbb{R}^P$
4. point-wise nonlinear transformation in lower-dimension, $\mathbb{R}^P \times \mathbb{R}^P$

The only constraint on this transformation is that the nonlinear transformation be differentiable. Note, the transformation defined above applies only for a two-layer network. In the general case the FNN can have any number of layers. Between Step 2 and Step 3 there may exist any number of linear projections, $\mathbb{R}^N \times \mathbb{R}^N$, coupled with point-wise nonlinear transformations, $\mathbb{R}^N \rightarrow \mathbb{R}^N$.

Gradient descent is the most widely applied method for training the FNN. Gradient descent is *supervised* learning, and, in the context of an FNN, this method is called backward propagation of errors, or *backpropagation* [65]. Theoretically, backpropagation is the FNN specific application of the derivative chain-rule. The requirement of the nonlinear transformation being differentiable ensures that the chain-rule can be iteratively applied to each step of the transformation. The trainable parameters of the first linear projection are the hidden weights, $W_{hid}$, and the trainable parameters of the second linear projection are the output weights, $W_{out}$.

Backpropagation is performed by computing the derivative of the FNN output's error with respect to the parameters, $W_{hid}$ and $W_{out}$, respectively. This derivative is interpreted as the the slope of the error surface in parameter space. The FNN parameters are modified by adjusting them in the direction of lower error, which is the descent direction of the slope.

A small step is taken in this direction, and the parameters are modified. This process is repeated iteratively until the gradient is sufficiently small (ideally zero), which indicates the model has reached the bottom of a basin in the error surface.

Despite its generality and well-defined, computationally-tractable training rule, the FNN possesses a significant weakness; the trainable parameter space is typically high-dimensional and contains many local optima. Another practical challenge is identification of training parameters that balance function approximation accuracy with generality. A large body of literature exists in identifying and resolving the nuances of neural network training, including, but not limited to: high-order derivatives [10, 52, 1], adaptive gradient descent methods [28, 19], statistical methods [14, 17], and nonlinear preprojection of the input data [53, 60, 42].

## 3.4 Recurrent Architectures

A mathematically well-defined way to represent arbitrarily long temporal features compactly (i.e., in closed form) is to represent them recursively. Of particular use is a class of nonlinear recurrences subsumed under the general description of recurrent neural networks (RNN). The RNN is, essentially, a feed-forward neural network in which the high-dimensional feature space (hidden state) of the previous prediction step is mapped onto itself to enhance input of the current prediction step, thereby defining the recurrence.

While structurally very similar to the FNN, the RNN functionality is governed by an entirely different mathematical realm—nonlinear dynamic systems. Self-reference (i.e., mapping the hidden state onto itself) adds representation capability. Unfortunately, it also introduces mathematical challenges well-known to the field of dynamic systems: bifurcations, instability, and chaos.

The remainder of this section will mathematically outline the standard fully recurrent modeling problem and then examine the most common training techniques. In the next section, an architecture will be introduced that avoids the problems of bifurcations and instability in the temporal recurrence. It is this architecture, the reservoir architecture, which will be used as the model of non-Markovian domains in this research.

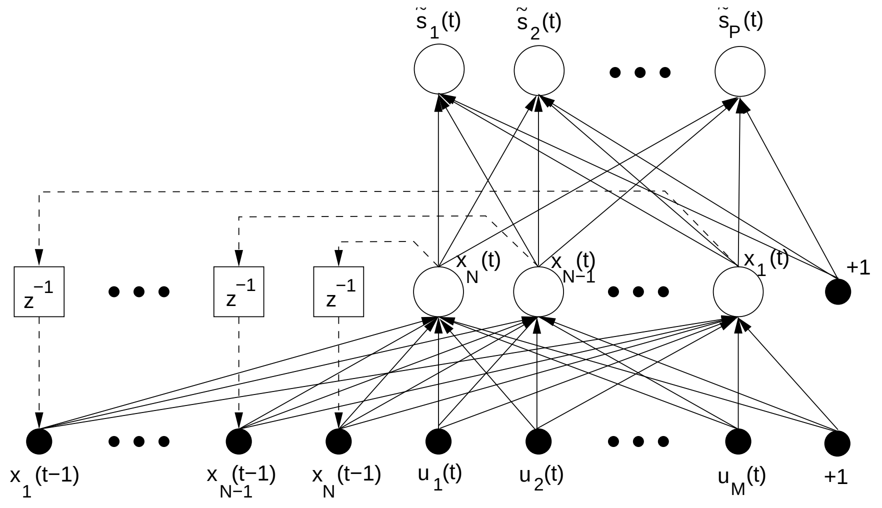

Figure 3.2: General schematic of the standard recurrent network model. Solid arrows indicate trainable weights.

## 3.4.1 Standard Recurrent Network Model

Many RNN architectures exist, but training methods are largely independent of the recurrent architecture. Therefore, theoretical descriptions of training methods are aided by a unified model. Consider a system where the true state is defined as vector $\mathbf{s}(t)$ of size $P$. Consider that an external influence vector $\mathbf{a}(t)$ of size $A$ acts on this system. The prototypical recurrent architecture for modeling this system is termed the standard recurrent network model (SRNM). The SRNM is a two-layer fully recurrent network mapping the input vector, $\mathbf{u}(t) = (\mathbf{s}(t), \mathbf{a}(t))$, of size $M = P + A$, onto the approximate system state, $\tilde{\mathbf{s}}(t)$. The SRNM possesses a recurrently connected hidden layer containing state $\mathbf{x}(t)$ of size $N$. The state of the output layer is identical to the approximate system state, $\tilde{\mathbf{s}}_t$. A depiction of the SRNM is provided in Figure 3.2.

The input state and hidden state of the SRNM are separate entities. When evaluating an RNN, however, the mathematics are best represented when the input and hidden states are concatenated along with a bias to form vector $\mathbf{y}(t)$ of size $N + M + 1$ such that:

$$\mathbf{y}(t) = (x_1(t-1), ..., x_{N-1}, x_N(t-1), u_1(t), u_2(t), ..., u_M(t), 1)^T$$

The recurrent topology is then defined by the matrix $\mathbf{W}_{hid}$, which is of size $(N+M+1) \times N$.

The output layer defined by matrix $\mathbf{W}_{out}$ of size $(N+1) \times P$ maps the vector $\mathbf{z}(t)$,

$$\mathbf{z}(t) = (\mathbf{x}(t), 1)^T \tag{3.4}$$

onto the approximate future system state, $\tilde{\mathbf{s}}(t)$. Thus, temporal evolution of the SRNM hidden and output states can be defined by the recurrences,

$$\begin{aligned} \mathbf{x}(t) &= f\left(\mathbf{W}_{hid}^T \cdot \mathbf{y}(t)\right) \\ \tilde{\mathbf{s}}(t) &= g\left(\mathbf{W}_{out}^T \cdot \mathbf{z}(t)\right) \end{aligned}$$

For all experiments in this research, $f(\cdot)$ is hyperbolic tangent (i.e., *tanh*) and $g(\cdot)$ is the identity function.

This standardized architecture was chosen to allow for nonlinear recurrent connectivity with bounded hidden state while permitting full dynamic range over the output state vector. This form largely mimics the *ultimate* architecture defined by Pedersen [58] and is analogous to the majority of RNN architectures found in the literature.

### 3.4.2 SRNM Training Problem

Almost invariably, recurrent network training minimizes the model's *generalized mean-squared cost*. This measurement is defined as the mean-squared cost over a training dataset that is temporally contiguous with but not included in the validation and testing datasets. The *cost function* is more than prediction error. The cost function, $\mathbf{J}(t)$, can be formulated in multiple ways, but a generic definition is

$$\mathbf{J}(t) = \frac{1}{2}|\mathbf{e}(t)|^2 + \mathbf{J}_{misc}(t), \tag{3.5}$$

where $\mathbf{e}(t)$ is defined in Equation 3.1 and $\mathbf{J}_{misc}$ is a miscellaneous cost term reserved for introducing additional penalty terms, such as the regularization penalty, into the total cost estimate. Recurrent networks train over an entire temporal trajectory of a system, $\mathbf{s}(t)$, $t = 1, 2, ..., T_{train}$ where $T_{train}$ is the size of the training dataset. The prediction error over an entire trajectory is the total cost, $J_{tot}$, defined as

$$\mathbf{J}_{tot} = \sum_{t=1}^{T_{train}} \mathbf{J}(t). \tag{3.6}$$

### 3.4.3 Gradient Training Methods

The success of gradient techniques for training traditional FNNs has been influential in their use for training RNNs. Gradient descent training of recurrent networks has been attempted in numerous forms:

- static recurrent weights with backpropagation, Jordan and Elman networks [38, 39, 18];
- approximate first derivative backpropagation, backpropagation through time [77];
- exact first order derivative backpropagation, real-time recurrent learning [81];
- exact second order derivative backpropagation, real-time recurrent learning [58]; and
- trajectory modeling, Atiya-Parlos recurrent learning [4, 68].

The application of gradient descent techniques to recurrent networks, however, has had limited success. While the technique has been demonstrated to work effectively on small, academic problems it has failed to scale to larger problems—the exception being back-propagation through time (in conjunction with EKF), which, in the company of expertly manipulated training data and parameter initializations, has been demonstrated on large, challenging real-world problem instances [20]. The reasons for limited RNN application are long convergence times, training instability, and sensitivity to training parameters.

The literature also contains work outlining progress in explaining the theoretical limitations of RNN gradient descent training, notably Amari [2] and the dissertation work of Pedersen [58], which detail the effects of ill-conditioning in RNN training. No technique however addresses the problem of bifurcations in the RNN parameter space. Gradient techniques are unlikely to work effectively in the *online* training of dynamic system models because the gradient gives no indication of the presence or proximity of bifurcations. Weight updates in the direction of the gradient can unwittingly push the RNN through a bifurcation. This particular weakness of RNN gradient training is known [23]. Bifurcation exposure is averted by avoiding the online training problem.

Minimizing generalized prediction error is the objective of RNN training. All gradient descent algorithms suffer from potential overfitting of the training data when cost incorporates only prediction error. Two widely known techniques exist for improving generalization of an RNN: *regularization* [24] and *pruning* [45].

*Regularization* defines the technique of modifying the cost function to include the magnitude of the parameter matrices. Nonzero parameters cause this magnitude to grow and are added to the squared prediction error as part of $\mathbf{J}_{misc}$ in Equation 3.5. By changing gradient descent to optimize over a prediction surface that is also a parameter surface, the regularization term provides a resistance to overfitting and undesirable complexity. Depending on the dominant cost term, the gradient direction of improvement will either minimize prediction error, minimize parameter magnitude, or a minimize a linear combination of both.

*Pruning* is the process of removing or "zeroing" weights from the recurrence portion of the RNN which have small saliency, or influence, on prediction training error. Pruning trades a small increase in prediction error for a large decrease in network complexity [45]. Pruning is performed by constructing a local approximation of the error function with respect to the parameters, computing the salience of each parameter based on the derivative of this approximation, and iteratively eliminating parameters having minimal influence on prediction until the desired trade-off is achieved.

The literature of RNN training via gradient descent is substantial. This summary has touched only the most well-known and prominent gradient methods. For an excellent overview, see [57]. Also, this summary has not addressed a significant technical debate within the RNN community concerning the "vanishing gradient" problem reported by Hochreiter. For a detailed examination of this problem, see [27, 26].

### 3.4.4 Statistical Training Methods

The Kalman filter [41] was designed to model any unknown process governed by a linear dynamical system in the presence of noise. The Kalman filter models this system by estimating the most likely hidden state, given the training data. This technique is extensible to a parameterized function approximation by assuming that the approximation's parameters comprise part of the state of the unknown process. The technique improves estimation of the internal state by maintaining covariance information on state element interactions. These interactions are related to higher order statistical information on the approximation's parameterized error surface.

The extended Kalman filter (EKF) is a nonlinear extension of the Kalman filter. The nonlinear portion is linearized locally in an additional step. Once linearized, the EKF is solved as a Kalman filter [83]. The extended Kalman filter has been shown to successfully train RNNs on very hard problems and is utilized in real-world industrial applications [20]. This technique was also shown to be related to the RTRL gradient technique [83, 82].

### 3.4.5 Sampling-based Training Methods

The Alopex algorithm [7, 8, 75, 17] is a general sampling-based optimization algorithm that can be used to optimize the parameters of a neural network. Alopex works by randomly perturbing model weights and associating these perturbations with increases or decreases in prediction error. Correlations between beneficial perturbations are used to modify the likelihood of future perturbations. An annealing schedule is used to slowly decrease perturbations over time. A recent modification to Alopex that incorporates particle filtering [25] has demonstrated significantly improved performance on a nonlinear non-Markovian, financial forecasting task. However, in spite of this advance, Alopex has not been demonstrated to be competitive with well-tuned gradient and statistical methods.

### 3.4.6 Search Methods

The use of genetic algorithms (GA) and evolutionary algorithms (EA) for constructing or optimizing networks is a recent but growing field of exploration. There exists evidence that in combination, evolution and learning can achieve performance greater than either individual technique.

An excellent general treatise of applying genetic algorithms to neural models is provided in [67]. An argument of the advantages of evolutionary algorithms over genetic algorithms is made in [3]. In both surveys, the case is made that search methods are favorable. The problem of competing conventions, multiple equivalent solutions induced by the symmetry of neural networks, however, makes genetic encoding of networks inefficient. No general mechanism for overcoming the competing conventions problem has been found and it remains the single largest theoretical barrier to GA implementation of training neural networks.

The specific use of evolutionary simulated search for application in reinforcement learning and control domains is well-posed by Whitley [78]. Wielend utilized genetic algorithms to train neural networks that solved the double pendulum swing-up and the jointed pendulum control problems [79, 80]. GAs were also used to achieve comparable performance to gradient methods for training neural networks to solve classic control problems [78]. The potential superiority of search to gradient methods has been posed for RNN training due, specifically, to the stability and convergence limitations of gradient methods in these domains.

## 3.5 Reservoir Architectures

The Echo State Network (ESN), and related Liquid State Machine, are models of temporal learning designed to avoid the slow convergence, computational complexity, stiffness, and instability of recurrent network learning algorithms [30]. ESNs rely on the inherent dynamics of a large, stochastically generated, sparsely connected recurrent structure in which many small, loosely-coupled dynamical systems interact with the input data to form a rich set of features. Interest in these systems has spawned a community of research covering the echo state network, liquid state machine, and a related technique, backpropagation-decorrelation [68, 69], bundled under a unified title, *reservoir computing.* With respect to non-Markovian reinforcement learning, what separates the reservoir architecture from other architectures is that it possess all three positive attributes: 1) ESN training is equally amenable to online and batch training, 2) the ESN cannot bifurcate, and 3) training is fast, efficient, and convergent in the least-squares sense.

The remainder of this section is organized as follows. First the mechanics ESN prediction and training are presented. Subsequent sections detail the state-of-the-art research conducted on the ESN architecture, including memory analysis and performance improvement.

### 3.5.1 Echo State Network Model

Consider a system where the true state is defined as vector $\mathbf{s}(t)$ of size $P$ and the external influence is a vector $\mathbf{a}(t)$ of size $A$. The ESN is a two-layer stochastically generated, sparsely connected recurrent network mapping the input vector,

$$\mathbf{u}(t) = (\mathbf{s}(t), \mathbf{a}(t), 1)^T \tag{3.7}$$

of size $M = P + A$ onto the approximate system state at the next time-step, $\tilde{\mathbf{s}}(t)$. The ESN possesses a recurrently connected hidden layer having state $\mathbf{x}(t)$ of dimension $N$. The state of the output layer is identical to the approximate system state, $\tilde{\mathbf{s}}(t)$. A depiction of the ESN architecture is given in Figure 3.3.

The ESN contains three separate connection topologies: input connections onto the hidden nodes, $\mathbf{W}_{in}$, of size $M \times N$; recurrent connections within the hidden layer, $\mathbf{W}_{hid}$, also termed the *reservoir*, of size $N \times N$; and the output mapping from the hidden state vector onto the approximate system state, $\mathbf{W}_{out}$. To improve performance, $\mathbf{W}_{out}$ maps an embellished hidden state vector, $\mathbf{z}(t)$, defined as,

$$\mathbf{z}(t) = (\mathbf{u}(t), \mathbf{x}(t))^T \tag{3.8}$$

of size $M + N + 1$, thereby giving $\mathbf{W}_{out}$ size $(M + N + 1) \times P$. Thus, the temporal evolution of the ESN is defined in recurrence relations,

$$\mathbf{x}(t) = f\left(\mathbf{W}_{in}^T \mathbf{u}(t) + \mathbf{W}_{hid}^T \mathbf{x}(t-1)\right) \tag{3.9}$$

$$\tilde{\mathbf{s}}(t) = g\left(\mathbf{W}_{out}^T \mathbf{z}(t)\right) \tag{3.10}$$

where activation function $f(\cdot)$ is typically *tanh* or a spiking neural network form [30] and $g(\cdot)$ is the identity function. The ESN training rule is defined as linear regression applied to the over-specified system of linear equations,

$$\mathbf{B}\mathbf{W}_{out} = \mathbf{S}. \tag{3.11}$$

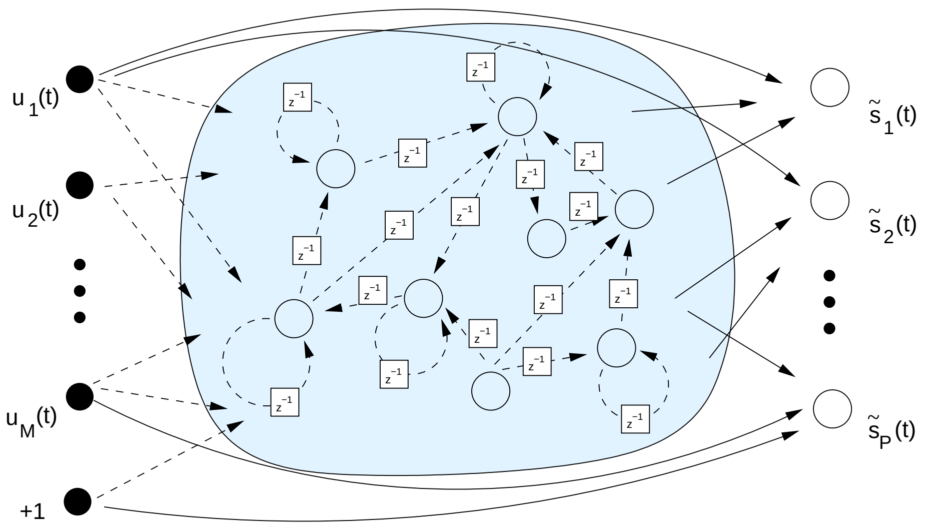

Figure 3.3: General schematic of an Echo State Network. Dashed arrows indicate static, stochastically-assigned weights. Solid arrows indicate weights trained via linear regression.

$\mathbf{B}$ is a matrix having $T_{train}$ rows where $T_{train}$ is the total number of training samples. Each row of $\mathbf{B}$ is the vector $\mathbf{z}(t)^T$. Each row of the constraint matrix, $\mathbf{S}$, is the true state vector $\mathbf{s}(t)$, $t = 1, ..., T_{train}$.

The ESN requires specification of three parameters, $N$, $\alpha$, $\rho$. As defined above, $N$ is the number of hidden states (i.e., echo states). The scaling factor, $\alpha$, when applied as a scalar multiple of $W_{hid}$ guarantees global stability [30] if,

$$\alpha < 1/ \mid \lambda_{max} \mid \tag{3.12}$$

where $\lambda_{max}$ is the maximum absolute eigenvalue of $\mathbf{W}_{hid}$. The density parameter, $\rho$, determines the percentage of nonzero entries in $W_{hid}$.

### 3.5.2 ESN Research: Recent Advances

The disadvantage of the ESN is exactly the attribute which engenders its strengths. The stochastically generated reservoir connection weights, $\mathbf{W}_{in}$ and $\mathbf{W}_{hid}$ are not adapted at any time. Therefore, if the reservoir does not satisfy the assumption that its dynamics are "rich enough" then the ESN will have difficulty modeling challenging dynamic systems. This weakness has been high-lighted both by critics and supporters [61, 30]. The current focus of ESN research has revolved around four directions: improved reservoir design, online adaptation of reservoir dynamics, analysis of reservoir computability, and practical application. This section will expand on these research directions.

Reservoir Design: Jaeger has been the primary proponent of hand designing reservoirs to match particular problem dynamics [30, 32]. He has developed a number of rules of thumb that are intuitive and easy to apply: tuning spectral radius to system dynamics, input and bias scaling weights to saturation characteristics, and nonlinear dynamic models to a target system's intrinsic time-scale. In a related work, Jaeger formulates the short-term memory capabilities of the ESN reservoir and suggests design rules-of-thumb for tuning the reservoir's memory-curve characteristics [31]. As practical proof of these simple techniques, independent researchers (including this author) have reproduced impressive published non-Markovian modeling results on very difficult problems including near "best known" results for closed-loop prediction of the Mackey-Glass time-series in both mildly ($\tau = 17$) and very chaotic ($\tau = 30$) regimes [30, 61].

Ozturk and Principe have taken an information theoretic approach to reservoir design. By linearizing the reservoir and examining poles, they use maximum entropy principles to uniformly distribute the poles within the unit circle in the complex plane, creating a quasi-orthogonal dynamic basis. In the absence of *a priori* system knowledge, a maximally decorrelated dynamic basis is the best that can be achieved [55, 56].

Adaptation of Reservoir Dynamics: ESN reservoir dynamics have been adapted both offline and online. The best example of offline adaptation was reservoir topology adaptation using next ascent local search [11]. The impact of this work was the demonstration of easily accessible and accurate gradient information within topologically local neighborhoods in the reservoir. More directly applicable to this research is the adaptive bias approach [56] proposed by Ozturk and Principe. The primary result of this work demonstrated that the magnitude of the constant input bias influences the effective (i.e., dynamic) spectral radius of the reservoir. Based on a correlation between spectral radius and the frequency of reservoir dynamics, input bias manipulation can be used to tune reservoir dynamics to the problem online.

Reservoir Computability: Recent work using the ESN as an analytical model has demonstrated the efficiency of performing predictive modeling tasks using random dynamic basis functions exhibiting "near chaotic" dynamics (i.e., bordering chaos just inside the ordered regime of the ordered-chaotic dynamic envelope) [6].

ESN Applications: The ESN architecture solves many training issues of the fully connected RNN. Solving these problems has made non-Markovian modeling easily accessible to a number of practical and theoretical applications including reinforcement learning [12], fast, effective nonlinear system identification [59, 76], cognitive modeling [63] and real-time mobile network modeling [33, 35].

## 3.6 Recurrent Actor-Critic Models in Discrete Space

The research considered in this proposal is partially motivated by prior research of Mizutani and Dreyfus [49, 50, 51] that attempted non-Markovian reinforcement learning under several simplifying assumptions: discrete space; finite, static time-dependencies; and model availability. This research successfully modeled, via the Elman network architecture, non-Markovian reinforcement learning of a shortest-path problem. The problem was formulated as the traversal of a binary tree, beginning at the root. The traversal was made non-Markovian artificially by penalizing specific multinode transition patterns. While the problem domain is excessively simple, this research simultaneously demonstrated three important components of non-Markovian reinforcement learning:

Representability: Both the actor and critic function were modeled via a fully recurrent Elman network trained via error backpropagation.

Accuracy: The actor and critic models were explicitly non-Markovian and the Elman network successfully represented these functions with very high accuracy.

Convergence: Under the static environment described by the short-path domain, the actor-critic method converged to policy and action-value function approximations that were near optimal.

## 3.7 Discussion

The requirements of non-Markovian reinforcement learning, outlined in Chapter 2, combined with a review of the broad literature of predictive modeling, direct attention to the ESN architecture as the most amenable to the task at hand. Based on theoretical properties, the ESN exhibits the stability necessary to serve as a non-Markovian model of reinforcement learning. Further, the ESN has demonstrated excellent learning performance on complex nonlinear prediction problems using a simple, convergent training mechanism. Open questions addressed in this research are the applicability of the ESN, and particularly its training mechanism, to nonstationary attractors. In the particular case of reinforcement learning, online interaction between the system and the model induce this nonstationarity. To identify the ESN's suitability in this environment, I first assesses the mobility of the reinforcement learning domains in the general sense. In Chapter 4, interactions between the reinforcement learning domain and the model of state-action value (i.e., critic) are studied for both local and nonlocal functions approximations. Building on this understanding, the feasibility of the ESN model is studied on both simple and complex non-Markovian reinforcement learning domains in Chapters 5 and 6, respectively.

Chapter 4

# Modeling Reinforcement Learning Domains as Dynamic Systems

This research approaches the reinforcement learning problem as a nonlinear, dynamic system modeling problem. When viewed as a dynamic rather than static system, interactions between temporal-difference updates, the environment, and the modeling architecture may be considered in ways not easily accessible to a statistical approach. Consideration of these interactions transforms the practical difficulties of modeling reinforcement learning domains from state-space sampling and convergence to those encountered when modeling dynamic systems—attractor mobility and bifurcations.

As an example, the Mountain Car Problem, a widely-known and well-understood reinforcement learning domain, is studied as a dynamic modeling problem. Approximations of the domain, realized in the form of policies, determine the attractor's structure. Variations in this approximation are described, culminating in the attractor of the near-optimal policy found via lookup table. With these relationships in mind, the dynamics of the learning process may be addressed. Contrasts are drawn between the learning dynamics of a highly local model, the lookup table, and a non-local approximation, the feedforward neural network. Once these dynamics are presented, the problem of attractor mobility may be framed. This problem is inherent to reinforcement learning. An experimental framework for testing mobility is presented and a study of attractor mobility in response to the architecture of the critic model is carried out. Ways of minimizing the impact of attractor mobility in reinforcement learning are proposed. These methods will be investigated experimentally in

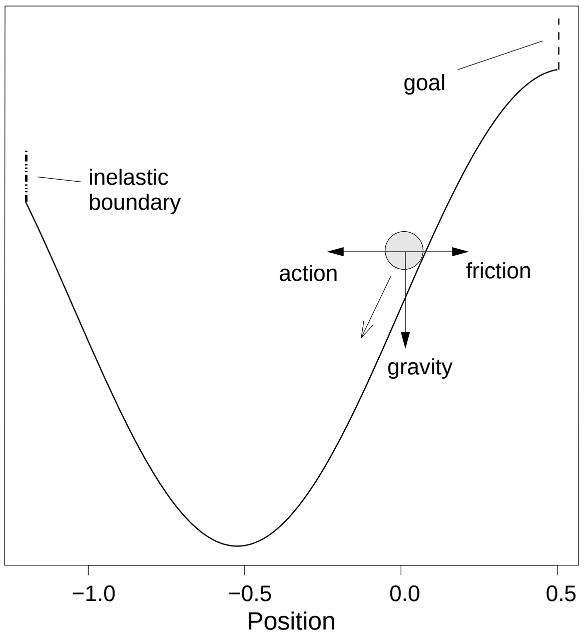

Figure 4.1: Diagram of the Mountain Car Problem.

Chapter 5 when the problem domain is expanded to include non-Markovian reinforcement learning.

## 4.1 Case Study: The Mountain Car Problem

The Mountain Car problem (MCP) is a second-order, nonlinear dynamic system with low-dimensional, continuous-valued state and action spaces [73]. In this section specifics of the problem dynamics and constraints are introduced. An approximate, near-optimal solution to the problem is presented and the dynamic structure of this solution is analyzed.

MCP is a one-dimensional abstraction of a simple environment—a car trapped within a valley. This system is depicted graphically in Figure 4.1. Starting from any initial state, the goal of the problem is to drive the car up the valley's side and escape in as few steps as possible. The challenge is that the car does not possess enough force to drive directly up the side. Instead, the car must rock back and forth along the bottom of the valley to build enough momentum to escape. The problem is typically complicated by the placement of a perfectly inelastic barrier halfway up one side of the valley. This barrier introduces a discontinuity in the physics of the problem. When this barrier exists, the goal of the problem is to drive the car out of the valley on the side opposite the barrier. Exact constraints of the MCP vary. The system used in this research is defined by the following differential equation,

$$\ddot{x} \quad = \quad -b\dot{x} + gm\cos(3x) + \frac{a\tau}{m}, \tag{4.1}$$

where the coefficient of friction $b = 0.3$, force of gravity, $g = 9.8$, car mass, $m = 0.2$, and maximum force, $\tau = 0.2$. The action, $a$, is drawn from the set $a = \{-1, 0, 1\}$. The inelastic boundary is defined as follows,

$$\dot{x} \quad = \quad \begin{cases} \dot{x}, & \text{if } x > -1.2; \\ 0.0, & \text{else.} \end{cases} \tag{4.2}$$

The goal state is defined as $x \geq 0.5$. The reward, $r$, is defined as

$$r \quad = \quad \begin{cases} -1, & \text{if } x > 0.5; \\ 0, & \text{else.} \end{cases} \tag{4.3}$$

## 4.2 Mobility in the Reinforcement Learning Domain

The reinforcement learning problem is a dynamic system. Unlike most dynamic systems, however, the underlying attractor of the reinforcement learning domain is inherently mobile, where mobility is defined as either bifurcation or displacement of the dynamic system's equilibria.

To define mobility formally two functions, $fp(\cdot)$ and $sc(\cdot)$, are presented which determine the existence of a fixed point and the stability class of a fixed point, respectively, for an input state, $\mathbf{s}$. Thus,

$$\begin{aligned} fp(\mathbf{s}) \quad &= \quad \begin{cases} 1, & \text{if } \dot{\mathbf{s}} = 0 \text{ and } \ddot{\mathbf{s}} = 0; \\ 0, & \text{else.} \end{cases} \\ sc(\mathbf{s}) \quad &= \quad \begin{cases} [sign(\frac{\partial(\dot{\mathbf{s}}+\Delta\dot{\mathbf{s}})}{\partial t}), sign(\frac{\partial(\dot{\mathbf{s}}+\Delta\dot{\mathbf{s}})}{\partial t})], & if\ fp(\mathbf{s}) = 1; \\ null, & \text{else.} \end{cases} \end{aligned}$$

Using these identifying functions, the current mobility of the system, $M_t$, can be defined as the sum of all mobility events, $m_s$, in the state-space, $S$, at time $t$, given by

$$\begin{aligned} M_t \quad &= \quad \sum_{\mathbf{s}_t \in S_t} m_{\mathbf{s}_t}, \text{ and} \\ m_{\mathbf{s}_t} \quad &= \quad \begin{cases} 1, & \text{if } fp(\mathbf{s}_t) \neq fp(\mathbf{s}_{t-1}) \\ & \text{else if } fp(\mathbf{s}_t) = fp(\mathbf{s}_{t-1}) \text{ and } sc(\mathbf{s}_t) \neq sc(\mathbf{s}_{t-1}) \\ 0, & \text{else.} \end{cases} \end{aligned}$$

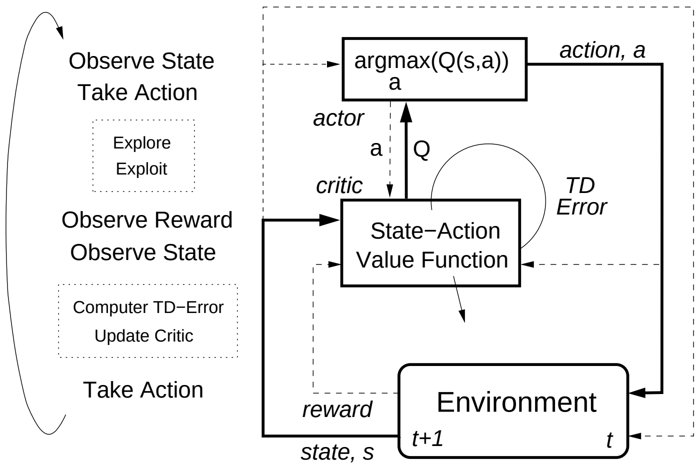

Figure 4.2: The recurrent pathway that is the source of attractor mobility in the reinforcement learning domain.

These definitions provide a course measure of changes occurring within the structure of a dynamic system. How does this relate to reinforcement learning?

The temporal interactions of reinforcement learning are presented graphically in Figure 4.2. This figure shows future state, $\mathbf{s}_{t+1}$, as a function of the current state, $\mathbf{s}_t$, and action, $a_t$. But the action, $a_t$, is a function of the state and the parameters of the $Q$-function, via the actions selected by the actor, in this case the function $\arg\max(a, Q(\mathbf{s}, a))$, also given in Equation 2.14. The action selected is guaranteed to be the optimal action, $a_t^*$, if $Q = Q^*$. Likewise, over a sequence of actions from initialization to goal, the policy $\pi$ is the optimal policy, $\pi^*$, if $Q = Q^*$. The attractor underlying the reinforcement learning domain is formed by the policy $\pi$ interacting with system dynamics, forming trajectories in state-space, by Equation 2.11. Thus, the attractor is static only if the policy $\pi$ is static, which by Equation 2.14, requires $Q$ to be unchanging.

Reinforcement learning attempts to minimize the error between $Q$ and $Q^*$ by modifying the parameters of the critic in the direction of the TD-error update, Equation 2.3. Examined in this way, attractor mobility in reinforcement learning is a direct result of approximation errors of Q. TD-error, $\delta_t$, must exist if $Q \neq Q^*$. If actively learning, then updating the critic, $Q_{t+1} = f(Q_t, \delta_t)$, updates the policy, $\pi_{t+1}$, which changes the system's underlying attractor. The result is attractor mobility.

In the dynamic system sense, reinforcement learning exhibits two properties: 1) during learning, parameters of the Q-function follow a trajectory through time, and 2) within any particular instance of time, the current manifestation of the reinforcement learning problem is defined by a dynamic system, having an underlying attractor.

### 4.2.1 Policy Driven Dynamics

The intent of this section is to demonstrate how reinforcement learning heuristics, used to guide Q-function approximations, induce mobility of the reinforcement learning domain's attractor. Before getting to the actual learning heuristic, several intermediate attractor configurations are presented that are likely to be visited during learning of the MCP. The attractors are presented in Figure 4.3. The attractors have been plotted by generating trajectories over predefined policies where the initial conditions are sampled on a discrete grid. For clarity, the gross attractor features have been highlighted to indicate equilibria and direction of flow. In practice the underlying attractors will exhibit much less homogeneous structure.

With no *a priori* knowledge, there exists no historic experience to build a Q-function approximation. The result is that no intelligent policy exists in this case. Two policy configurations well-suited to this case are provided in Figure 4.3(a) and (b), the *no action* and *random action* policies, respectively. These attractors exhibit similar fixed point structure, a stable spiral. From almost every initial configuration of position and velocity, the state space evolves in a trajectory that approaches the stable fixed point at the center of the spiral. This is a configuration typical of a conserved system where energy dissipates. The qualifier "*almost* every states evolves to the equilibrium" is justified because there exist several initial conditions that lead directly to the goal state. In these cases, the position is far enough advanced to the goal and the velocity has a high enough positive value such that a trajectory reaching the goal is inevitable without action. It should also be pointed out for the *random action* policy that all policies of this type will be different. Random sampling of actions during system evolution, however, does not greatly change the global dynamic structure.

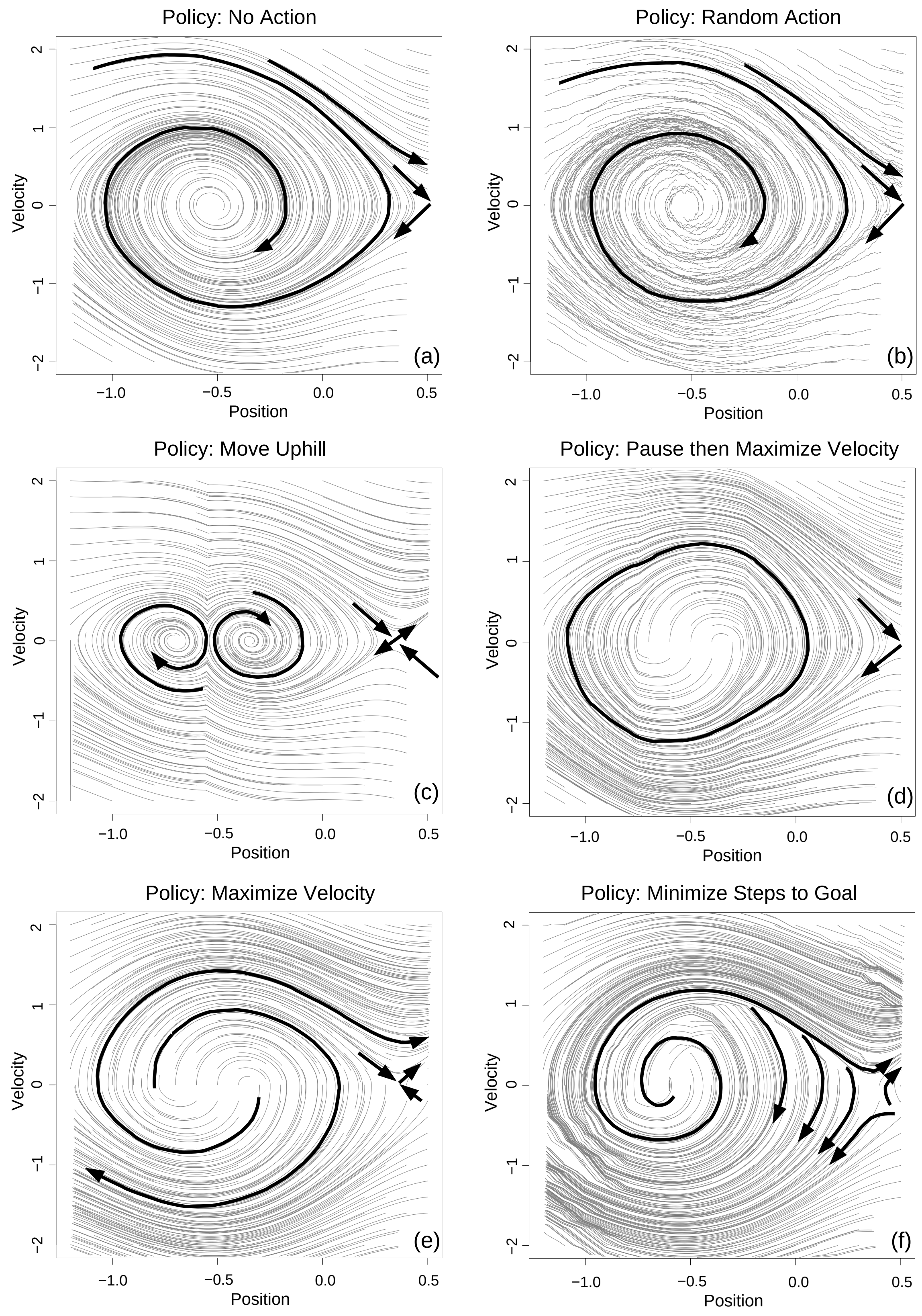

Figure 4.3: Examples of policy induced attractor structures. Changes among these policies would constitute attractor mobility.

Now consider policies which do admit *a priori* information. These policies are based on knowledge of the MCP and its solution. The first of these policies, shown in Figure 4.3(c), propels the agent uphill at all times. The attractor underlying the dynamic system formed by this rule exhibits two stable spirals on either side of the floor of the valley of the MCP. This result is intuitive. Without alternating between positive and negative actions, the agent pushes in a single direction until the slope of the valley becomes too great and the force of gravity overpowers the agent's force. The agent, still pushing, redescends toward the valley floor until the force is sufficient to accelerate upwards. This process will repeat, until the initial momentum contained within the agent is dissipated by the frictional coefficient of the problem, $b$. A second feature of this attractor is the appearance of a saddle point equilibrium at $x \approx 0.4$. The saddle point marks a separation of the state-space in which the agent's force, combined with initial positive velocity cannot overcome the dissipative force of gravity and friction.

Figure 4.3(d), is an attractor structure that will be revisited in this research repeatedly. The policy exhibited here is one of velocity maximization. However, the maximization of velocity only occurs when $-0.75 \leq x \leq -0.25$. Outside of this interval the agent takes no action. This policy simulates a commonly observed intermediate Q-function configuration observed during on-line reinforcement learning in which the agent develops a highly efficient escape policy that suddenly degenerates. Dynamically the attractor in this configuration exhibits a stable limit cycle encircling a unstable spiral centered on the MCP's valley floor. The stability of the limit cycle arises from the agent being repelled from the fixed point at the valley floor but becoming "confused" when its position moves outside the interval defined above. This confusion causes the agent to redescend into the valley. The limit cycle forms, approximately, at edges of the interval defining the transition from velocity maximization to no-operation. The structure of this attractor must be compared to that of velocity maximization, depicted in Figure 4.3(e). Removal of the "confusion boundary" allows the agent to maximize velocity throughout the entire state-space. This change in policy has a dramatic effect on the structure of the underlying attractor. In this case, the limit cycle has vanished and the unstable spiral dominates a majority of the state space.

The saddle point has again appeared in the attractor dividing the physical limits of the system under near-optimal policy decisions.

Finally, the attractor underlying the optimally solved MCP is depicted in Figure 4.3(f). Note the similarity of this attractor to that of velocity maximization. The primary difference between these attractors is the structure of the unstable spiral along a trajectory connecting the fixed point at $x \approx -0.6$ with the saddle point. Trajectories falling inside this spiral curve away. This nonlinear structure is the result of the minimum path criteria of reinforcement learning. Trajectories in the optimal solution must be as short as possible. Therefore, velocity maximization, while a good policy, is sub-optimal. The goal of MCP is to maximize velocity *toward* the goal.

### 4.2.2 Learning with Minimum Mobility

When reinforcement learning is viewed as the evolution of an attractor through time, it is possible to describe the ideal learning trajectory. Ideal, not only in creating an optimal policy, but rather, creating an optimal policy while satisfying useful constraints, such as minimum mobility. Using structural analysis of the system's underlying attractor it is possible to quantify the learning trajectory. For the purposes of this research the best reinforcement learning trajectory is one that finds an optimal policy with the minimum change to the system's underlying dynamics, a *minimum mobility pathway*, between the initial policy, built without *a priori* knowledge, and the final, optimized policy.

At the beginning of this section, the formal definition of mobility was presented. This definition expressed mobility as changes to observed structure of the attractor underlying the reinforcement learning domain. While this provides a means of measuring changes to the attractor, a deeper understanding of fundamentals of attractor mobility is useful.

The structure of a nonlinear dynamic system may be approximated by linearizing the system locally about its fixed points and computing the eigenvalues of this system. The structural characteristics of the attractor about the fixed points may be locally determined by these fixed points. For example, fixed points in a 2-dimensional system, having eigenvalues $\lambda_1$ and $\lambda_2$, are characterized by the left hand diagram in Figure 4.4, where the x-axis is

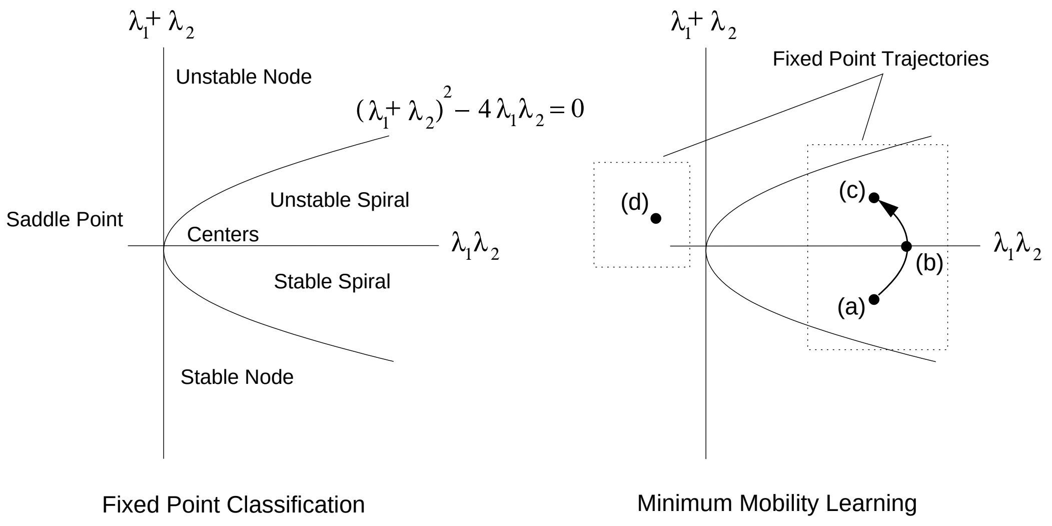

Figure 4.4: Theoretical motivation for minimum attractor mobility in a 2-dimensional dynamic space.

the *determinant*, $\lambda_1\lambda_2$, and the y-axis is the trace, $\lambda_1 + \lambda_2$.

The righthand plot of Figure 4.4 shows an idealized MCP trajectory moving through this space, joining the initial attractor structure, the stable spiral at point(a), with the final attractor structure, the unstable spiral at point (c), coinciding with a saddle point at point (d). Viewed in eigenvalue space the learning trajectory of MCP makes sense. The system's eigenvalues have values mapping to the initial attractor, point (a), and then transform to values mapping onto the final attractor structure (points c and d). The shortest trajectory between points (a) and (c) in this space must cross at some point (b). With eigenvalues mapping onto point (b), the system will include a *center* fixed point, a class of point of which limit cycles are a part.

The rigor of attractor structure in 2-dimensional eigenspace provides conceptual understanding of the types of attractor structures expected as a reinforcement learning domain evolves. These concepts will appear throughout this research as interactions between function approximations and reinforcement learning are investigated.

Further, use of the Echo State Network as a non-Markovian model is motivated by demonstrating that the trajectory-based feature space of the dynamic basis exhibits less mobility than static state-space mappings. Demonstrating this, and understanding why dynamic trajectories serve as stable, robust feature spaces is the unifying theme between this chapter and Chapters 5 and 6.

## 4.3 Approximate Optimal Solution

A solution to the MCP can be approximated, near-optimally, using a lookup table of the critic's state-action-value surface. The critic was modeled via a lookup table of 3-dimensions. The table was indexed according to the following discretizations of the state-action space: action, $a$, was drawn from the set $\{-1, 0, 1\}$; position, $x$, was discretized on the range $[-1.2, 0.5]$ at intervals of 0.1; velocity, $\dot{x}$, was discretized on the range $[-2.7, 2.7]$ at intervals of 0.1. The total number of parameters in this function approximation was $3 \times 18 \times 55 = 2970$. The lookup table was trained via SARSA, specified by Algorithm 1, given in Appendix A.

Thirty state-action-value function approximations were learned. Learning terminated when the function approximation converged, defined as the absolute value of the most recent 1000 TD-errors (i.e., Equation 2.3) summing to a value less than the $threshold = 0.001$. The tables were averaged for smoothness, creating the approximation of the optimal state-action-value function.

## 4.4 Structural Analysis of the Mountain Car Attractor

In Chapter 2, the reinforcement learning problem was derived such that Equations 2.15—2.18 completely describe the domain's dynamics in terms of state, $\mathbf{s}$. In the case of the MCP, the state is a vector, $\mathbf{s} = \{x, \dot{x}\}$. This state-space can be used to describe the MCP's underlying attractor, assuming a policy of $argmax(a, Q(\mathbf{s}, a)$, as depicted in Figure 4.3(f). During a brief description of nonlinear dynamic systems in Chapter 1, an attractor was characterized in terms of the location and stability of its equilibria. In this section I analyze the equilibria of the MCP's underlying attractor. This analysis will later be used as a reference for the transient dynamics observed while learning in a reinforcement learning domain.

The MCP state is comprised of both position, $x$, and velocity, $\dot{x}$. However, when examining the location and stability of equilibria in this domain, $\dot{x} = 0$ can be assumed because the time-derivative of position is velocity. Plotting the MCP's attractor in one dimension at $\dot{x} = 0$ yields Figure 4.5. Equilibria of the attractor exist at points where the velocity derivative (i.e., acceleration) is zero, depicted as a dashed line. These fixed points have been labeled with X-symbols.

The MCP attractor exhibits three fixed points at $x \approx \{-0.545, 0.388, 0.418\}$. Further, the stability of fixed points can be approximated by observing the sign of the velocity derivative when I perturb the velocity slightly away from zero, $\pm\epsilon$, and recompute the velocity derivative. The signs of the velocity derivative at small perturbations (i.e., $\epsilon = \Delta\dot{x} = 0.1$) are plotted with respect to their fixed points in Figure 4.5. The signs of $\pm\epsilon$ perturbations can take on four combinations, having various dynamic significance:

$$sign\left(-\epsilon, \epsilon\right) = \begin{cases} \{+,+\} \text{ or } \{-,-\}, & \text{semi-stable fixed point,} \\ \{+,-\}, & \text{stable fixed point, and} \\ \{-,+\}, & \text{unstable fixed point.} \end{cases} \tag{4.4}$$

Stable fixed points are attractive from both positive and negative perturbations of velocity. This implies that the trajectory within the state-space is decelerating and approaching this point—much like a basin of attraction. In contrast, unstable fixed point indicate that state-space trajectories accelerate away from the fixed point, akin to a topological peak. Semi-stable fixed points indicate simultaneous acceleration and deceleration on opposing velocity half-planes. These types of fixed points are physically difficult to imagine but could be considered small plateaus in a physical system with little momentum and high friction.

The advantage of viewing a reinforcement learning domain in terms of fixed point analysis is that it allows for concise interpretation of the interactions between the agent and the system's dynamics. Given our knowledge of the agent, the valley's minima is known to be at $x = -0.6$, which is the epicenter of any rocking motion to escape. The nearby opposing semi-stable fixed points at $x = 0.4$ form a saddle. This point marks the force limitations of the agent. The center of the saddle is a point where the force of the agent is sufficient to exactly counter-balance the effects of gravity with zero acceleration—a highly unstable equilibrium point, but one with two clear diverging trajectories seen in Figure 4.3(f). In situations where the agent's velocity is less than zero, it must redescend into the valley to achieve escape velocity. At velocities above zero, sufficient force exists to drive straight out of the valley. This saddle point marks the true decision boundary of the MCP. The agent must learn this location exactly to optimize its escape. Any misapplication of force near this point could lead to highly sub-optimal control.

Approximating the number, location, and stability of fixed points of the MCP's underlying attractor provides a quantitative measure of the attractor's structure. As stated in Chapter 1, bifurcations are changes in the number or stability of fixed points in an attractor, and represent fundamental changes to the dynamics of the system. Therefore, the ability to identify bifurcations provides a means of understanding how the underlying attractor, and, therefore, the solution space of a reinforcement learning domain, is changing through time. These calculations form one component of attractor mobility analysis, and they provide a means of measuring how state-action-value function approximation architectures behave during reinforcement learning.

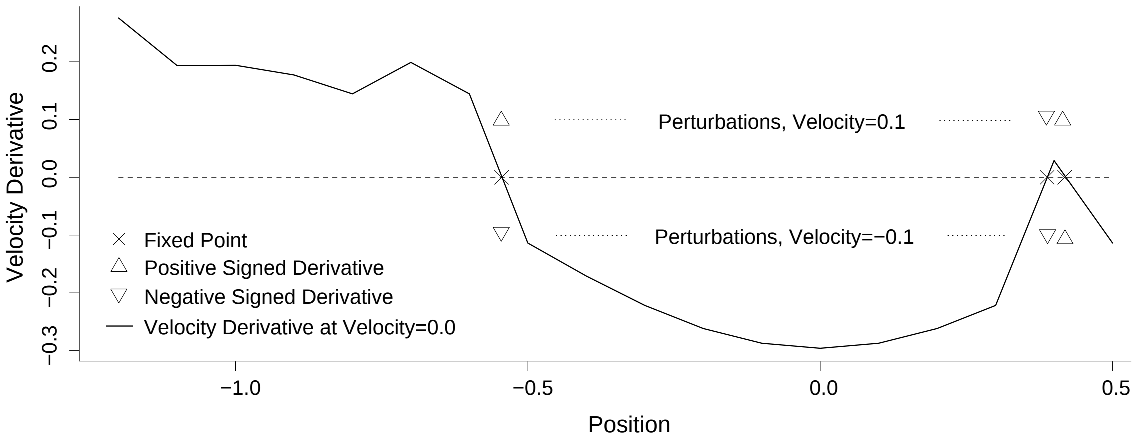

Figure 4.5: Fixed point and stability analysis of a lookup table based approximation to the Mountain Car Problem.

## 4.5 Attractor Mobility during Reinforcement Learning

Analysis of the underlying attractor of a reinforcement learning problem is useful as a means of interpretation—understanding how the system and control decisions interact. Reinforcement learning, however, is not simply a technique for finding a system's underlying attractor, but rather, the process of evolving the underlying attractor given no *a priori* information—a substantially more challenging problem. This problem is difficult in that the state-action-value function, which defines the attractor via the $argmax(a)$ function, is unknown, and, therefore, intermediate decisions (i.e., action selections) are likely to be suboptimal.

The learning of the state-action-value function requires the parameters of the function approximation to be adapted online—each intermediate approximation of the state-action-value function also forms a dynamic system having an underlying attractor, resulting in underlying attractor mobility.

While known theoretically, mobility in the underlying attractor of a reinforcement learning problem is not well-studied. In this section I identify both bifurcations and mobility in the MCP reinforcement learning domain and relate these features to learning performance. The information gained from this study is valuable in that it provides a means of representing and comparing, quantitatively, the underlying structures of reinforcement learning domains. It also provides a pathway to discovering the dynamics of online reinforcement learning with function approximations, which is an important step in designing learning architectures that function well in reinforcement learning domains.

### 4.5.1 Attractor Mobility in an Explicitly Local Model

To illustrate attractor mobility, I compute the location and stability of fixed points in the attractor described via lookup table approximation of the state-action-value function. This lookup table is trained using Algorithm 1 in Appendix A. These fixed points, labeled by stability-class, are plotted with respect to the number of training steps. Fixed points are computed and plotted every 1000 training steps. The results of this learning experiment for the first $10^6$ learning steps are presented in Figure 4.6.

Two important learning artifacts are obvious from Figure 4.6. First, local update of the table lookup model does not eliminate attractor mobility—bifurcations and displacements of fixed points is prevalent throughout the early stages of learning. Second, both the stability class and locality of the fixed points stabilize over time. Attractor mobility is driven by the

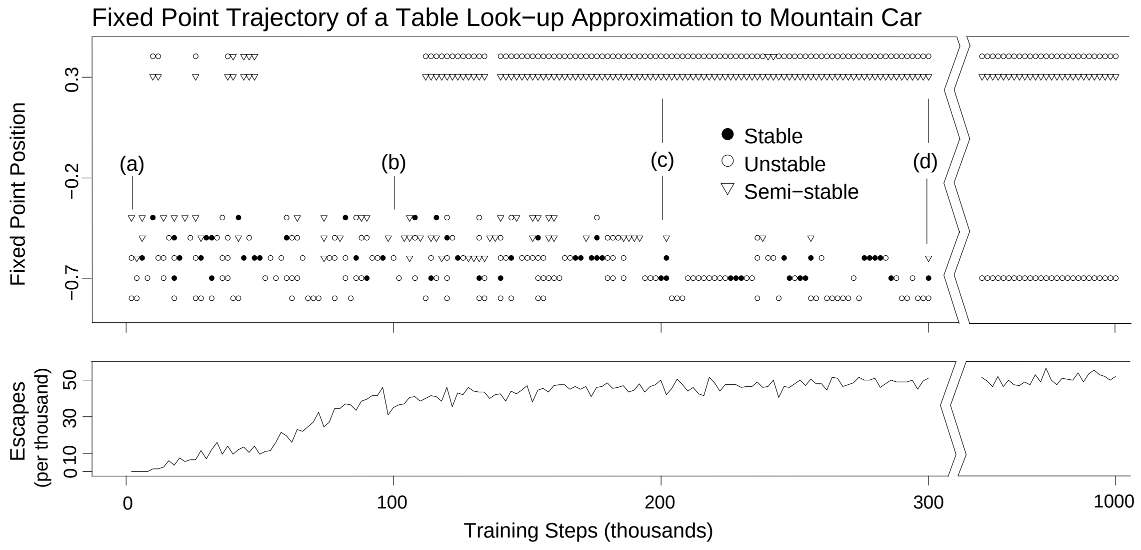

Figure 4.6: Bifurcation diagram of a lookup table based approximation to the Mountain Car Problem during learning. Fixed point based representation of attractor structure is highly local, however, emergence of the qualitative attractor features is visible through time. The attractor structure after after $10^4$, $10^5$, $2 \times 10^5$, and $3 \times 10^5$ updates, was explored as trajectories in state-space in Figure 4.6, corresponding to points (a—d).

impact of TD-error updates on the function approximation, which is in turn dictated by the optimality of the state-action-value surface. Parameters of a function approximating a surface which is close to optimal should be adapted very little. This minimizes, but does not omit, the approximation's exposure to bifurcations.

It can be argued that local updates to the parameterized approximation of the state-action-value function do not prevent non-local changes in the dynamics of the reinforcement learning domain. Non-local updates have been called the weakness of the neural network function approximations in reinforcement learning [44, 62]. The use of lookup tables with highly-localized Q updates does reduce the non-local dynamic impact of local updates, but it does not omit them. The impact is a matter of degree. The key feature of the lookup table is that experiences are modeled *exactly* at the local level. Every experience the agent receives is incorporated into the state-action-value surface only at the precise locality of approximation. Thus, the impact of a local approximation can be global with respect to the attractor, but remains local with respect to the state-action-value surface.

It would be useful to understand attractor mobility of the MCP with respect to a

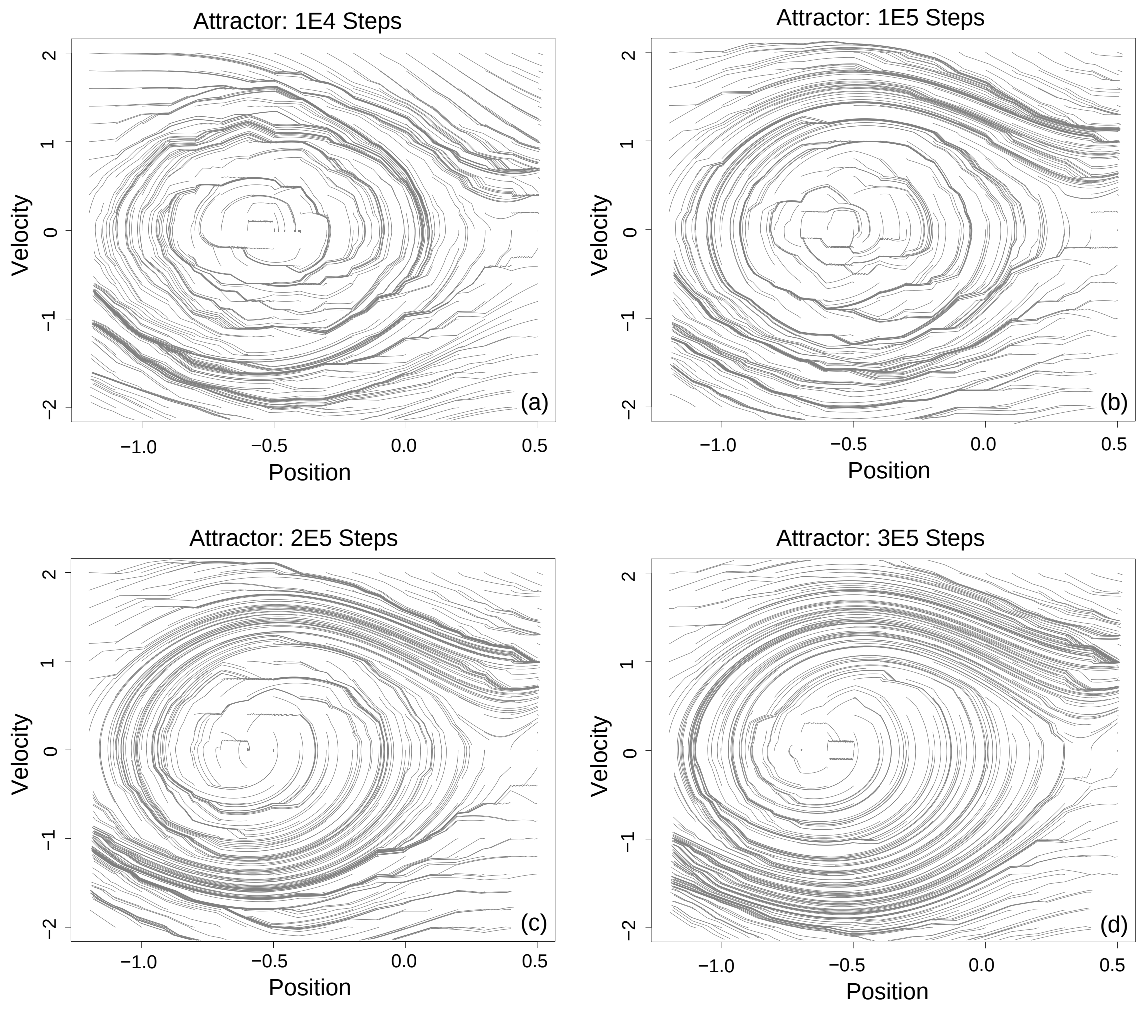

Figure 4.7: Example MCP attractors constructed from learning interactions between the system dynamics and the lookup table function approximation.

function approximation with a non-local update and relate this mobility to learning performance. Quantitative measurement of attractor mobility and its performance implications are presented in the next section.

### 4.5.2 Attractor Mobility in a Non-local Model

Compared with a lookup table approximation, the temporal development of the state-action-value function can differ substantially when using a generalized, parameterized non-local function approximation, such as a neural network. The most significant difficulty in learning with this type of model is the loss of long-term memory in local approximations. In a neural network, each point of new information interacts, potentially, with every parameter in the approximation. This interaction arises from the complete connectivity of the neural network model. During both feedforward and backpropagation phases, the input and error, respectively, are combined with every weight of the network. In the case of online learning, such as a reinforcement learning domain, locality of information is learned through temporal repetition of dispersed state-space information. To achieve this dispersion in control problems, randomness is added to the policy.

In this example I model the MCP with a neural network having 50 hidden units plus a constant bias. The internal nonlinear logistic function is *tanh* and the output layer is linear. The input domain is the complete state-action space, $\mathbf{u} = \{\dot{x}, x, a\}$, and the output domain is the state-action-value, $Q$. Thus, this approximation has $3 \times 51 = 153$ trainable hidden weights and $51 \times 1 = 51$ trainable output weights for a total of 204 parameters. All parameters are initially drawn uniformly from a range of $[-0.1, 0.1]$. The parameters are then adapted online via backpropagation of errors with fixed learning rates of 0.05 and 0.001 for the hidden and output layers, respectively. Probability of selecting a random action varied throughout learning according to a prescribed schedule. The probabilities $S = [0.8, 0.5, 0.1, 0.01, 0.0]$ were drawn in order every $25 \times 10^3$ steps[1] with an additional 1000 steps at $S = 0.8$ preceding the start of the schedule. Greedy actions were selected via

$$a = \arg\max_a Q(\mathbf{s}, a),\ \ a \in \{-1, 0, 1\}.$$

The initial position of the car was sampled uniformly from the range, $x_0 = [-1.2, 0.5]$, and initial velocity, $\dot{x}_0 = 0$, was always 0. The neural network was trained via SARSA, according to Algorithm 2, given in Appendix A. Figure 4.8 shows the position and stability-class of the fixed points approximated for a single MCP attractor modeled via neural network. The fixed points were calculated every 1000 training steps.

The speed of convergence of neural network approximation is quite good. Within just $200 \times 10^3$ learning steps, the attractor is very near its approximate local optimum, as measured by escapes per 1000 steps. The number of bifurcations observed in the neural

[1]In Algorithm 1, $maxsteps = 5000$ and $R = 5$

network is also less than the lookup table at equivalent levels of experiences, but this is clearly related to the smoothness of the approximation—the locality of the lookup table causes, potentially, many local changes in the velocity derivative at $\dot{x} = 0$. Each weight in the lookup table at $\dot{x} = 0$ is trained independently and may present a jagged surface, shown in Figures 4.7(a),(b), and (c).

The non-local impact of bifurcations in either case, however cannot be distinguished. The trend in this learning example is that the neural network possesses less noise in it's description of the attractor compared with the lookup table and that it achieves an equilibrium at an equally fast rate.

Two other features of Figure 4.8 to point out are: 1) the inverse correlation between randomness of policy selection and the number of escapes and 2) the temporal coincidence of escape performance with step-change decreases in the probability of random action. Both of these features are expected. Exploration should correlate with attractor mobility because it constitutes and explicit divergence from the optimal policy, $\pi^*$. Likewise, increasing exploitation of the learned attractor should increase performance if the attractor was correctly learned. However, large increases in performance suggest that the learning schedule was not ideally constructed for this learning trial because the percentage of exploration was artificially decreasing performance in portions of the space well-learned.

### 4.5.3 Limit Cycles in Mountain Car

Figures 4.9(a—f) depict snapshots of the MCP attractor during the learning trial presented in Figure 4.8. These snapshots were chosen because they highlight a problem common in neural network approximation of MCP, oscillation between unstable spiral and limit cycle structure of the fixed point modeling the valley floor. Figures 4.9(a—c) depict the early formation of the unstable spiral structure at $x \approx -0.4$. This sequence of events follows the expected attractor mobility pathway in 2-dimensional space, depicted in Figure 4.4. At some point in this sequence of attractor structures, the eigenvalue configuration of the model crosses from the 4th to the 1st quadrant of the *determinant-trace* space. Once the unstable spiral has formed, returning to the stable spiral configuration is not desirable. In

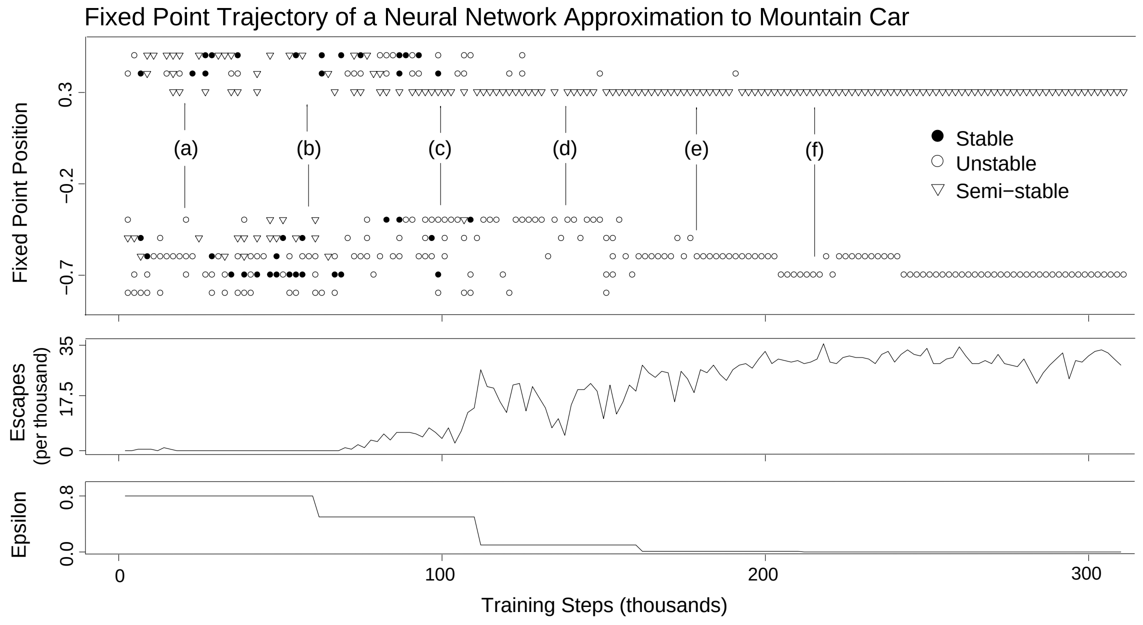

Figure 4.8: Bifurcation diagram of a feed-forward neural network approximation to the Mountain Car Problem. The attractor's structure was explored after $20 \times 10^3$, $6 \times 10^4$, $1 \times 10^5$, $1.4 \times 10^5$, $1.8 \times 10^5$, and $2.2 \times 10^5$ learning steps, indicated by points (a—f) in the figure and explored as state-space trajectories in Figure 4.9(a—f), respectively.

actual learning trials, the attractor is often seen changing from the unstable spiral structure to that of a stable spiral or limit cycle structure. In fact, this process occurs frequently during learning. Figures 4.9(c—e), depict the intermediate formation of a limit cycle, plot (e), between two unstable spiral configurations, plots (a) and (f).

To understand how and why these spurious intermediate structures occur, I want to consider the attractor from a different perspective. Using the fixed point structure of the attractor of the optimal-policy as a guide for partitioning the attractor's trajectories we can gain fundamental understanding of this spurious problem. To do this I connect the unstable spiral minimal pathway to the goal from the lowest value of Q to the saddle point. I further partition the state-space using the saddle point's structure[2]. I then label opposing sides of these boundaries. The result of this partitioning process is depicted graphically in Figure 4.10. The partitioning of the state-space in this way allows the entire reinforcement

[2]The saddle point discriminates the nearby vector field into quadrants defined by the eigenvectors of the linearized system.

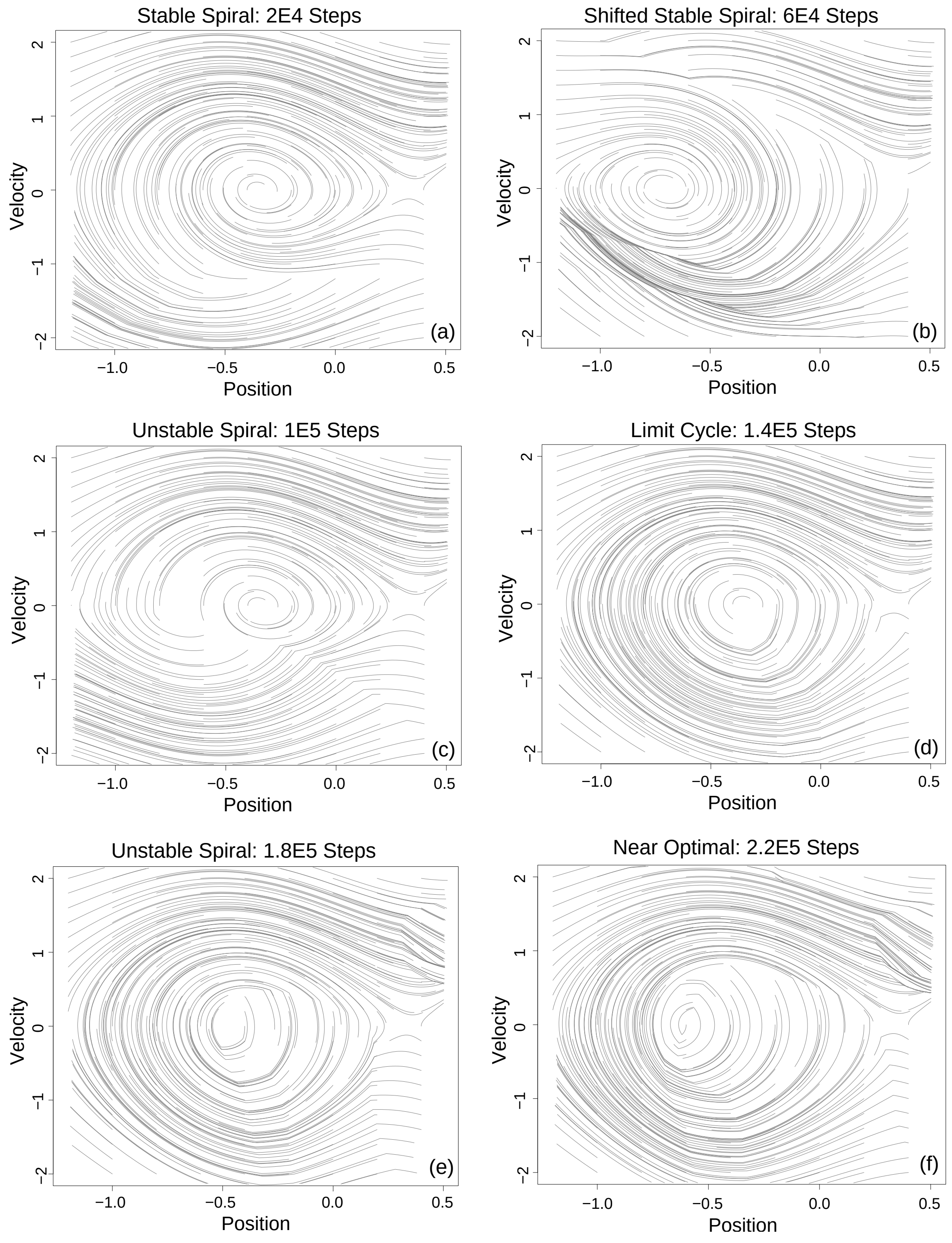

Figure 4.9: Example MCP attractors constructed from learning interactions between the system dynamics and the neural network function approximation.

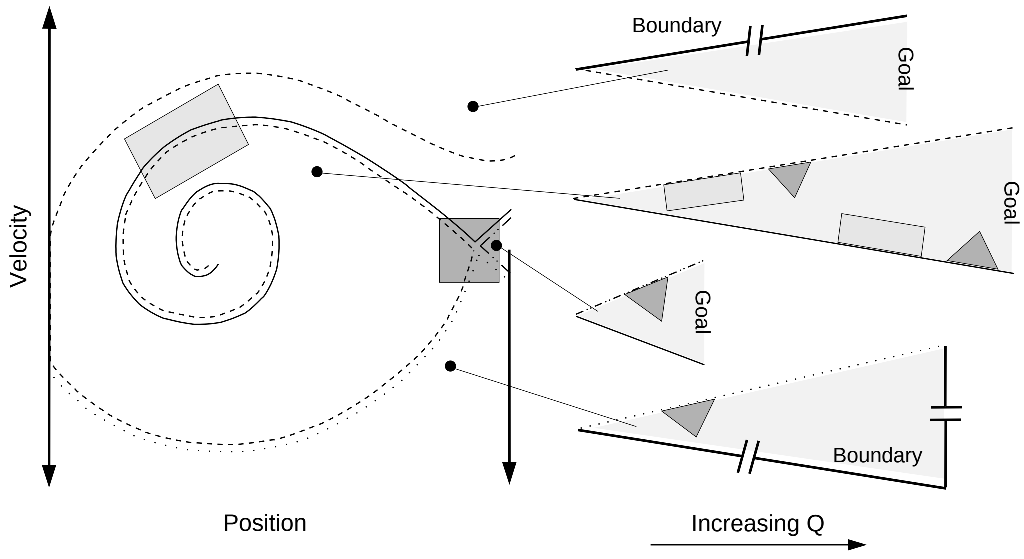

Figure 4.10: Decomposition of the MCP attractor into a set of linear structures. The linear structures are partitioned in state space by the MCP attractor's fixed point structure.

learning problem to be unrolled into a set of four quasi-linear trajectories, which describe pathways of maximal gradient ascent of the Q-surface. The equilibria of the attractor, literally, form the structural boundaries of this intrinsic state space.

The formation and disappearance of limit cycles in MCP is related to this intrinsic state-space. Because the quasi-linear space is the direction of gradient ascent, opposing ends of this space differ significantly in Q-value. When this space is coiled up into the complete state-space, the left plot in Figure 4.10, points having vastly different Q-value often lie next to each other, particularly along the curve joining the two fixed points. However, in the intrinsic space of the MCP, these point lie far apart. This property of MCP is depicted by the two regions shaded gray in Figure 4.10. In state-space, these shaded regions are contiguous. In the quasi-linear space, the regions are fractured. This illustrates the temporal relationships among points in these regions—points close together in state-space are temporally separated by the structure of the escape trajectories.

Non-local function approximations attempting to model Q-values along the boundaries of the intrinsic space will tend to bleed approximation across the line. These misapproximations induce the agent to make suboptimal decisions during the escape process. These suboptimal decision lead the agent to lose momentum and fail to escape, forming a cycle of the state-space, as shown in the sequence of Figures 4.9(c) and (d). This cycle is reinforced by the fact that the agent took a greedy decision, but the underlying approximation failed to represent this decision. Therefore the cycle is reinforced in the approximation, likely inducing further cycling of the state-space. The result is the transition of the unstable spiral to that of a limit cycle. If this process occurs sufficiently early in the learning schedule then exploration, fortunately, allows the cycle to be broken and the spiral to be reestablished. Representing the problem in state-space, rather than intrinsic space, is a motivating factor in the use of a fundamentally dynamic feature space, described in Chapter 5.

## 4.6 Analysis of Learning via Dynamic Features

Another question of interest is whether the dynamic temporal evolution of the attractor underlying the MCP determines learning performance. To aide in this study, I generate 1000 learning trials of MCP using the same parameters as those used to generate Figure 4.8. For each trial, fixed point locations, fixed point stability classes, and the number of escapes were recorded every 1000 learning steps. Each trial was run for $155 \times 10^3$ training steps. The performance of a trial was measured as the total number of escapes achieved during learning steps $101 \times 10^3$—$155 \times 10^3$. Analysis of the distribution of performance over the trials indicated that a natural break between successful trials and unsuccessful trials occurs at 1500 total escapes. This threshold was used to divide the trials into two classes in which the number of unsuccessful trials accounted for 13.6% of the total.

This experiment is intended to address two questions. First, can the success of a trial be predicted early in the learning process from the development of the attractor? Can the structural features relevant to distinguishing successful from unsuccessful trials be determined? As a corollary to this question, can the role of stochasticity in the determination of success be characterized?

To construct a classifier, the fixed point structure of individual learning trials was embedded in time. First, the dimensionality of the state space was reduced by eliminating positions which contain no fixed points. This reduced the dimensionality of the state-space from 18 points to 9 points. Once reduced, the fixed point structure was concatenated, sequentially, at each time-step. K-means clustering, $k = 2$, was then used to group the embeddings into two classes based on spatio-temporal structure of the fixed points. This embedding and clustering procedure was repeated for learning steps $10 \times 10^3$—$100 \times 10^3$. Once clustered, the labels assigned to successful and unsuccessful trials were used to determine the percentage of true positive classifications of each trial type. These percentages were calculated for each embedding length, resulting in the data presented in Figure 4.11.

The results of Figure 4.11 show that an embedding length of $38 \times 10^3$ learning steps, the classification accuracy rises from random (i.e., $\approx$ 50%) to non-random levels. For unsuccessful trials, classification accuracy peaks at approximately 83% and $41 \times 10^3$ learning steps. Classification accuracy of successful trials rises to 93% by $45 \times 10^3$ and improves gradually to 99% by $100 \times 10^3$. These results provide evidence that the most significant temporal changes in the attractor's fixed point structure occur slightly before 40E3 learning steps and that these changes are, essentially, all-or-nothing. Once the attractor evolves along one of two trajectories, this evolution is deterministic with respect to the success or failure of the learning trial.

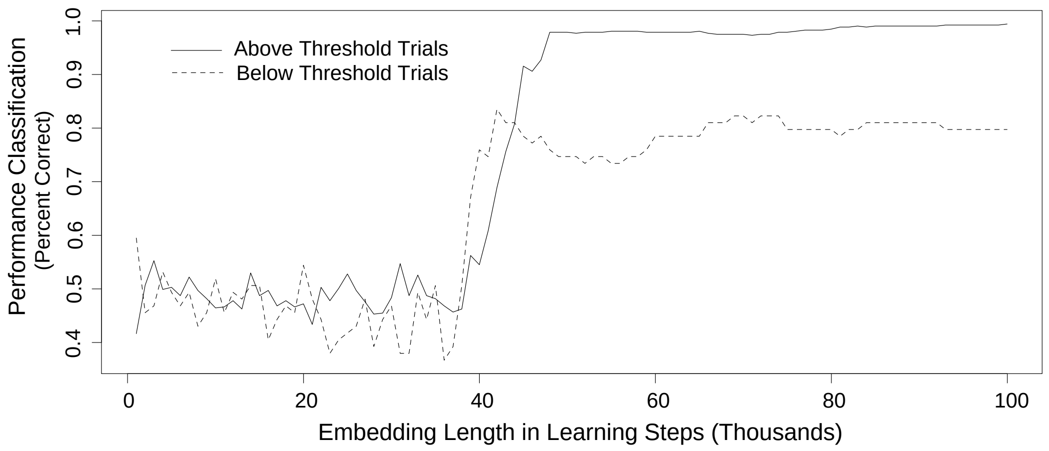

Figure 4.11: Fixed point feature space prediction of learning success. 1000 learning trials were separated into two classes, successful and unsuccessful. The threshold for learning success was defined to be $\geq 1500$ total escapes occurring in learning steps $101 \times 10^3$—$155 \times 10^3$. Fixed point structure of the trials was embedded in time and classified with $K-means$ cluster, $k = 2$. At an embedding length of $38 \times 10^3$ learning steps, the classification accuracy rises from random (i.e., $\approx$ 50%) to non-random levels. For unsuccessful trials, classification accuracy peaks at approximately 83% and $41 \times 10^3$ learning steps. Classification accuracy of successful trials rises to 93% by $45 \times 10^3$ and improves gradually to 99% by $100 \times 10^3$.

A secondary observation of Figure 4.11 is that unsuccessful trials are more difficult to predict from the spatio-temporal structure of their fixed points than successful trials. This difference indicates that successful trials are structurally homogeneous, while the spatio-temporal structure of unsuccessful trials are relatively less similar, and are, therefore, less cleanly modeled by the assumptions of K-means.

If possible, it would be insightful to understand exactly what spatio-temporal fixed point structure is exhibited by the two classes of learning trials and if this structure makes sense from a dynamic perspective of learning. To identify this structure principal component analysis (PCA) was performed on the classification dataset described above. However, PCA of the whole dataset only provides a picture of spatio-temporal structure, not causality. Instead, PCA was performed on three datasets: the subset of unsuccessful trials; the subset of successful trials; and the entire dataset. A summary of the results of this analysis is presented in Figure 4.12.

PCA is a statistical analysis method that decomposes a dataset from a coordinate system, or basis, derived from measurements (i.e., the dataset) into a coordinate system derived from the differences and similarities within the dataset—an intrinsic basis. Specifically, PCA decomposes the dataset into a set of ordered eigenvectors which define orthogonal directions of variance in the data, and a set of eigenvalues, also ordered, corresponding to the amount of variance in the data along each of the eigenvectors' directions. Figure 4.12(a), depicts the first principal component of unsuccessful trials, Figure 4.12(b) represents the first component of successful trials, and Figure 4.12(c) depicts the first four components of all trials, ordered clockwise by variance starting from the top-left. The y-axis of these figures is the reduced state-space (i.e., the location and stability class) of the fixed points constructed from nonzero rows of the original fixed point space as presented in Figures 4.6 and 4.8.

The results of Figure 4.12 suggest that the two largest components of variance in the data result from the variance contributed by the unsuccessful and successful trials, respectively. The first component of data subset of unsuccessful trials is nearly identical to the first component of the total dataset. Less obvious, the first component of the data subset of successful trials is also very similar to the second principal component of the total dataset. The sign of the direction is opposite, indicated by the opposite shades of gray, but this implies that it is a similar direction of variation.

Therefore, the dataset is structurally divided between learning trajectories that are structurally unsuccessful and those that are successful with heavy variance among the second class. Within the first principal components of the respective subsets of data, there also exists evidence of why the two distinct classes of attractor form, and this evidence is rooted in the trajectory of the attractor's dynamic components. Inspection of the first principal component of the unsuccessful trials indicates variance in the region of the saddle point ($x = 0.2$) at approximately $38 \times 10^3$ learning steps. This feature corresponds, temporally, with an increase in classification accuracy. Compare this spatio-temporal feature with analogous sections of Figure 4.12(b). The black band of Figure 4.12(a) is noticeably absent from Figure 4.12(b). Because it is known that high-quality solutions of the Mountain Car problem contain fixed point structure at approximately this position, evident from fixed point analysis of the near optimal attractor in Figure 4.3(f), and that there exists almost zero variance in this feature in the successful trials, it can be concluded that the black band in this region of Figure 4.12(a) is evidence of the absence or incorrect stability class of the saddle point in many, but not all, unsuccessful learning trials. It is very likely, therefore, that a significant component in the failure of learning in Mountain Car can be attributed to the loss of saddle point structure in the attractor at approximately $38 \times 10^3$ learning steps. This would explain, possibly, the entire classification of successful components and a large portion of unsuccessful trials. It is difficult, however, to explain why the remaining 17% of unsuccessful trials cannot be classified. In this case, these trials would be those which have saddle point structure at $x = 0.2$ yet are still unsuccessful.

One possible explanation of how misclassified unsuccessful trials can be resolved is by the light and dark gray bands at $x = [-0.8, -0.6]$ in Figure 4.12(a). In nearly all cases of unsuccessful learning, a stable fixed point would be expected at $x = -0.6$. However, visual inspection of the fixed point structure of unsuccessful trials indicates that this is not always true. In fact, several unsuccessful trials have *unstable* fixed points in this region, very much like the successful trials. An example of this artifact is Figure 4.9(d). While counter-intuitive, this behavior can be explained by the locality of fixed point stability analysis. This analysis does not consider the possibility of a stable limit cycle encircling the fixed point. In this scenario, the fixed point could evaluate as unstable, but the global behavior with respect to this location is stable (i.e., attractive). The limit cycle, while not correctly identified, is functioning much like a stable fixed point in these unsuccessful trials.

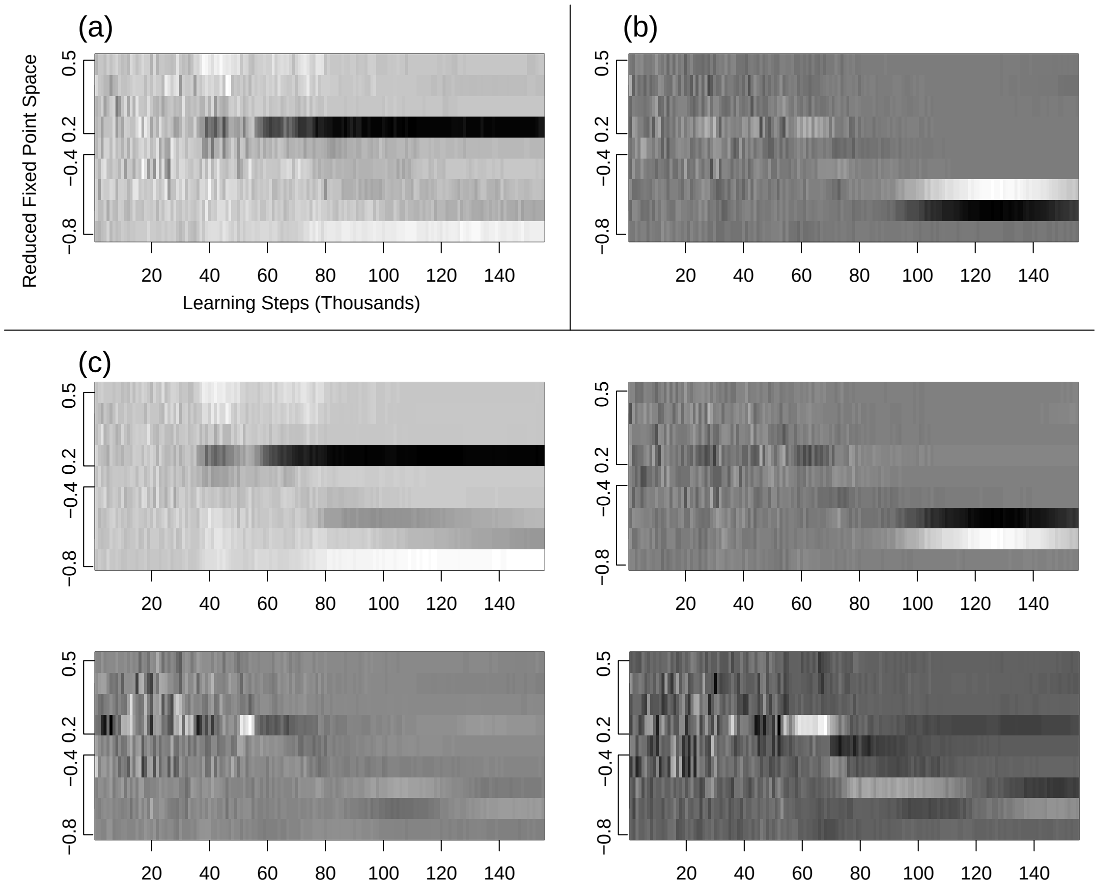

Figure 4.12: Principal components analysis of fixed point features of the Mountain Car Problem. (a) First principal component of unsuccessful learning trials. (b) First principal component of successful learning trials. (c) First four principal components of all trials, ordered clock-wise in decreasing variance starting from the top-left.

## 4.7 Discussion

Framing reinforcement learning as a dynamic system facilitates construction of spatio-temporal structure within the learning architecture, which explains, with high accuracy, learning performance. Principal component analysis of the temporal fixed point features indicates that success or failure of the learning process is rooted in the creation or destruction of key dynamic features of the learning domain's underlying attractor. The results of this analysis also indicates that, while the structure of the attractor determines learning, the likelihood of a modeling architecture evolving the necessary structure is stochastic.

The structural significance of the underlying attractor and the mobility of the attractor during learning indicate that a state-space representation of the state-action-value function is unsuitable. Ideally, reinforcement learning would be modeled in a space directly related to the structural components involved in learning—features of the underlying attractor. With higher-level features, such as fixed point structures commonly found in nonlinear dynamic systems, the complexity of the state-action-value space could be drastically reduced. The use of *options* [**?**], conceptually, is a step in this direction.

Attractor mobility also motivates the problem of modeling reinforcement learning to one of population dynamics. Experimental evidence suggests that a percentage of attractors trained on the problem will successful evolve the necessary structures to achieve learning. Therefore, constructing a learning agent from a population of models, each used as a voting element in a winner-takes-all selection of intermediate actions, would increase the probability that intermediate actions are selected based on dynamic features conducive to learning. This is a way of reducing the impact of attractor mobility and bifurcations on learning success. In theory, these agents could maintain sufficient diversity in the parameter space of a learning architecture such that if one or more bifurcations were to occur in the minority of models, the correct attractor structure could still be learned. This technique would be essential to on-line reinforcement learning.

Finally, the most important result of this analysis is the impact attractor mobility has on the probability of obtaining good learning solutions. This is true even when the complete system state-space is known. For incomplete state architectures, however, the outcome is potentially more dire, as was discussed for state-space representations in MCP inducing limit cycles during learning. Recurrent architectures possess inherent parameterized dynamic systems which are subject to bifurcation. The existence of attractor mobility dictates that the parameter space of the model will be subject to perturbations, which in turn exposes these parameterized dynamic architectures to bifurcations. The potential stability consequences for attractor mobility require consideration of incomplete state-space architectures which minimize the impact of mobility on learning.

# Chapter 5

# Modeling Mountain Car Problem via Echo State Network

Chapter 4 studied the dynamics of reinforcement learning when modeled via local and non-local function approximations in complete, Markovian state-space. The results of this study were threefold: 1) the attractors underlying reinforcement learning domains are highly mobile, 2) mobility is largely a result of exploration early in learning and misapproximation of Q during later stages of learning, and 3) the evolution of an attractor when modeled via non-local function approximation influences long-term learning performance.

Is attractor mobility avoidable? Particularly, does attractor mobility occur in dynamic feature spaces, such as ESNs? These are interesting questions because dynamic feature spaces are constructed from the system's intrinsic dynamics. Beyond the minimum necessary to evolve the initial attractor to the final learned attractor, based on the results of Chapter 4, attractor mobility is largely dependent on the structure of the feature space. If the feature space is the intrinsic feature space of the problem, mobility is expected to be minimal.

Another question often raised in the use of fixed dynamic bases is complexity issues stemming from a lack of adaptation. With respect to reinforcement learning, this would require demonstration that the basis is rich enough to represent the reinforcement learning domain's attractor throughout the learning trajectory from initialization to convergence. Also, if the dynamic basis is suitable for representing reinforcement dynamics, it would be expected that the learning performance would accelerate compared to static, non-local approximations. Accelerated learning is expected because the target domain projects linearly onto the dynamic basis and does not require adaptation of the features.

The goal of this chapter is to study modeling performance of the ESN in the MCP domain under both low and high speed learning schedules. To achieve this goal, first, a brief tutorial on practical ESN functionality is provided. The construction and behavior of the ESN implementation of the dynamic basis with respect to reinforcement learning is described. An example of MCP attractor dynamics is provided when the MCP domain is modeled via ESN. Finally, the performance of the ESN in learning the MCP compared to traditional Markovian architectures is analyzed. The analysis focuses on the ESN's representations of the reinforcement learning domain and provides guidance as to where the architectures strengths and weaknesses exist.

## 5.1 Design and Training Considerations in Reinforcement Learning

The principle difficulty in using the ESN architecture effectively is construction of a suitable reservoir that well-represents the dynamics of the problem at hand. This problem is well-known, encompassing both the reservoir richness problem [30, 56, 61, 11, 13], as well as the vagaries of reservoir design [30, 32, 36]. In the past, the ESN has been applied to applications in complex, nonlinear system identification—modeling static attractors.

For the purposes of this research, however, the interest is that of constructing a reservoir that well-represents a reinforcement learning domain, which, as described in Chapter 4, exhibits mobility. This section provides a practical framework for understanding, designing, and implementing an ESN for this task. First, basic ESN behavior and function is discussed. Building on this description, the remainder of the section details the design and training of the ESN in reinforcement learning domains.

### 5.1.1 Echo States: A Tutorial

The ESN is a dynamic basis. In short, the ESN projects a time-series into a high-dimensional space onto which future predictions of the time-series may be mapped, linearly, by the solution of an over-specified system of linear equations, given in Equation 3.11. A conceptual barrier to the use of the Echo State Network as a non-Markovian state-space approximation is proper understanding of what role the ESN architecture has in forming this system of equations. While the recurrent equations of the ESN are formally presented in Chapter 3, real ESN intuition may be found in a practical example.

Figure 5.1 depicts, temporally, the evolution of ESN internal state to external stimulation. As its name implies, the ESN "echoes" information about inputs through time. Differences between input sequences result in differences between the ESN echoes. When multiple inputs are presented to the ESN in sequence, the echoes of the various inputs interact nonlinearly. Step-by-step description of this behavior is as follows.

Figure 5.1 depicts the response of two identical ESNs under differing external input sequences. Initially, the internal state of the ESN is $\mathbf{0}$. Without external stimuli, the internal states of both ESNs remain silent. After two seconds of simulation time a small, positive external stimulus is input to each ESN. Conceptually, we can imagine this pulse input as a stone dropping into a pool of liquid. This external stimulus perturbs the ESNs internal state, much like ripples forming on the surface of perturbed liquid. Because the ESN is updated deterministically, Equation 3.9, both ESNs exhibit identical responses to identical inputs. After ten seconds of simulation a second external stimulus is applied to each ESN. The top ESN in Figure 5.1 receives a positively valued stimulus. The bottom ESN receives an negatively valued stimulus of the same magnitude. In response to the two external stimuli, the internal states of the two ESNs diverge through time. Between 10 and 40 simulation seconds, the internal states of the ESNs both slowly settle back to near zero.

The example given above is designed to highlight several features of ESN behavior: 1) temporal evolution of the internal state is deterministic, 2) temporal evolution of the internal state is unique for a unique input sequence (i.e., ESN dynamics are point-wise separable), and 3) temporal evolution of the internal state converges to zero without external stimulation (i.e., ESN dynamics are stable). Combined, these properties allow the ESN to act as a dynamic basis, a space on which a dynamic system may be mapped, linearly. How is this done? A dynamic system that is a function of the input sequences in Figure 5.2 may be modeled by assigning weights to each element of the ESN's internal state. The

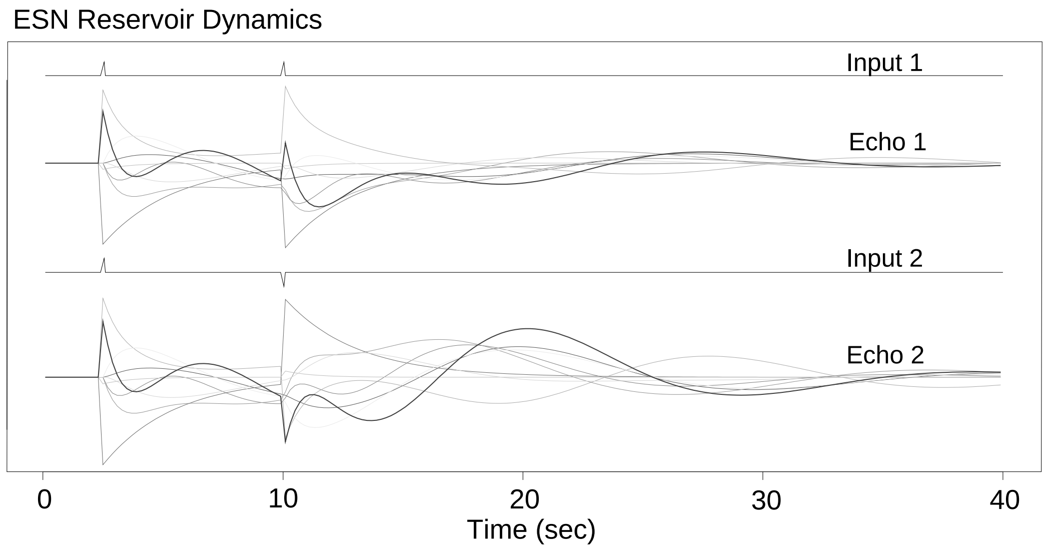

Figure 5.1: Example dynamic response from identical reservoirs under two unique input sequences.

time-series of the underlying system may then be reconstructed by the weighted summation of the echo states trajectory. The purpose of Equation 3.11 is to determine the weights that reconstruct the target system optimally in the least squares error sense.

### 5.1.2 Designing a Dynamic Basis for Reinforcement Learning

The difference in building a basis for a traditional dynamic system and that of a reinforcement learning domain is rooted in interactions between the action, system dynamics, and the sum of future rewards, $Q$, as given in Equations 2.15–2.18. Unlike nonlinear system identification tasks that map a current state, or current state and action pair, onto future states, the dynamic system of the reinforcement learning domain maps the state-action pair onto an approximate sum of expected future rewards signals.

This difference has several consequences. First, and most important, is that the prediction output influences the selection of the action received as an input, as in Equation 2.5. Equilibria within the reinforcement learning domain result from the function approximation of future rewards, which directly influence system dynamics. This is a significant difference from system identification in which the model passively observes the system's dynamics and searches for parameter values that best mimic these dynamics.

Another consequence of the reinforcement learning problem is that there exists an inherent, dynamic asymmetry to the target function—summation of future rewards. Therefore, the dynamic structure of the system and the target may be very different. An example of this difference is observed in MCP. When solved optimally, the summation of expected future reward signals is a monotonically increasing function along a trajectory, whereas, the system's dynamics are largely periodic.

An additional concern arising in a goal-directed problem domain, particularly that of MCP which is a path minimization problem, is that achieving the goal induces a reset of the system state to some random initialization. There is no obvious way of dealing with a system reset with respect to the internal dynamics of the ESN. For example, if the internal dynamics of the ESN are reset along with the system state, then the ESN's internal dynamics will require several iterations to suppress transient contextual signals resulting from this restart. In essence, a restart of the ESN's internal state destroys the quasi-Markovian state representation maintained in the reservoir's dynamics.

The final and most difficulty piece of this process is to consider flexibility of representations between the beginning and end of learning. The attractor of a reinforcement learning domain changes during learning. The echo state space itself must also be able to alter it's representation of the attractor through time. This is challenging because the final attractor structure will, in general, not be known *a priori*.

### 5.1.3 Reservoir Details

Reservoirs are stochastically generated, random nonlinear projections of the system's dynamics. As described in Chapter 3, the distribution of ESN features are determined by the ESN's density, spectral radius, and transfer function. These interactions have been explored previously for stationary attractors [56, 30, 47, 11]. The knowledge transfer between stationary attractors and nonstationary attractors, however, has not been studied. Without a literature to guide design decisions, the design focused on manipulating the reservoir's high-level parameters to satisfy four criteria in decreasing order of importance: 1) system dynamics, 2) reward signal structure, 3) horizon length, and 4) attractor mobility.

Based on these criteria and a suite of preliminary parameter studies, the ESN architecture tasked with learning the MCP was constructed according to the following parameters. The input vector was non-Markovian, $\mathbf{u} = \{x, b\}$, where $x$ is position and $b = 0.2$ is the constant bias. The reservoir state, $\mathbf{x}$, was of size $N = 100$. The input mapping, $\mathbf{W}_{in}$, density was 0.5, supplied with nonzero values sampled uniformly on the range $[-0.14, 0.14]$. The reservoir mapping, $\mathbf{W}_{res}$, had a density of 0.04 nonzero values sampled uniformly from the set $\{-1, 1\}$. $\mathbf{W}_{res}$ was normalized by the absolute value of its maximum eigenvalue and then scaled by $\alpha = 0.9$. No noise was added to the reservoir state during computation. The system is entirely feed-forward. Without feedback of the current prediction of $Q$, it is difficult to construct successive predictions without a large body of asymmetric echoes. Asymmetry in the ESN is difficult to generate due to periodicity of the echo states—highly asymmetric echoes are possible, but do not occur with sufficient frequency. To facilitate asymmetry, the squares of the echoes were added to the embellished echo state, such that $\mathbf{z} = \{\mathbf{u}, \mathbf{x}, \mathbf{x}^2\}$. This is a common augmentation of the basic ESN model [33, 32].

### 5.1.4 The Spiking Neural Network Model

One decision of utmost importance in designing ESN dynamics is the nonlinear transfer function. Static parameterized function approximations generally utilize *tanh* or *sigmoid*. These functions exhibit properties well-suited to these architectures. When employed as the transfer function in the ESN, however, the dynamics of the reservoir tend toward high frequency oscillations. To ameliorate this problem, the spiking neural network model has been employed [30, 36] having the following mathematical form,

$$\dot{\mathbf{x}} \quad = \quad \frac{1}{C}\left(-a\mathbf{x} + \tanh\left(\mathbf{W}_{in}\mathbf{u} + \mathbf{W}_{res}\mathbf{x}\right)\right), \tag{5.1}$$

where $C > 0$ is the reservoir's time constant and $a$ is the leak rate of the individual neural unit. Integrating numerically via Euler's method produces,

$$\mathbf{x}_{t+1} \quad = \quad \left(1 - \frac{a\Delta t}{C}\right)\mathbf{x}_t + \frac{\Delta t}{C}\tanh\left(\mathbf{W}_{in}\mathbf{u} + \mathbf{W}_{res}\mathbf{x}_t\right). \tag{5.2}$$

The spiking neuron model mimics some of the dynamic behaviors of more sophisticated biologically inspired models while remaining mathematically simple [30, 36]. An advantage of this model is that it incorporates memory into the individual neural elements. The neural elements individually maintain historical context, much like a momentum, and are resistant to high-frequency perturbations. With proper selection of constants $C$ and $a$ this transfer model facilitates smooth dynamic feature representation of the systems studied in this work. For the experiments described in this chapter, the time constant was assigned $C = 0.44$, the leak rate was $a = 0.9$, and the neural model integration time-step was set to that of the problem integration time-step, $\Delta t = 0.1$. These parameters are in line with reservoirs used to model problems of similar dynamics and are known to be robust to variation [30, 36].

### 5.1.5 Observing the Echoes Dynamically

The design process of the echoes was carried out primarily by trial and error, starting from a template reservoir, described above, that exhibited good dynamics for a challenging stationary attractor system identification task [30]. Once constructed, the parameters of the system may be examined with respect to the following tunable parameters:

Saturation of the echo states: The input and recurrent weight sizes can be reduced to lessen the possibility of input summations saturating the tanh nonlinearity of Equation 5.2. This can also be achieved by reducing the spectral radius of $\mathbf{W}_{res}$ to increase the echo decay rate.

Echo time-scales: Adjusting the neural leak rate $a$ and time constant $C$ varies the time constants of echo states.

Non-linear augmentation of the ESN: The asymmetry of the sum of expected future reward signals can be addressed at the level of the embellished echo state. Non-linear transformations of the echoes, such as the squares, concatenated to the base embellished echo state, $\mathbf{z}$, can achieve dynamics well-suited to modeled a monotonically increasing nonlinear function.

Transients: The echo state was reset to $\mathbf{0}$ upon restart or initialization of the system state. Transient effects can be controlled through manipulation of the reservoir's spectral radius. The spectral radius of a reservoir is positively correlated with the number of iterations it will exhibit transients.

Echo diversity: Maximizing the rank of the echoes (i.e., the number of linearly independent echo trajectories) best ensures diversity without *a priori* knowledge. Sparseness in the reservoir, controlled through small values of $\rho$, facilitates full rank echoes [11].

Using these tools, the reservoir may be tuned such that it is well-suited to linear mapping of the learning problem. This is a difficult, ill-defined problem, which is why trial and error was employed to find suitable reservoir parameters for these learning experiments. Figure 5.2(a) depicts the evolution the three most highly weighted echoes.[1] The approximate sum of future rewards, predicted by the reservoir during this learning trial, are presented in Figure 5.2(b) as reference. While examining this figure, remember that the temporal evolution of the echoes is influenced by the static recurrent connectivity of the reservoir, the linear readout weights (used to predict $Q$ *and* select actions), and the constraints of the transfer function.

Figure 5.2(a) illustrates the temporal aspect of attractor mobility within the echo states. During the earliest phase of learning, without much experience, the ESN echoes are disorganized. The disorganization is the result of the large percentage of random actions injected into the reservoir due to state-space exploration, as well as misapproximation of $Q$ resulting in poor action selection. As a result, the readout is poorly mapped onto the Q-function, and, therefore, only the most gross predictions are possible, as evident in the leftmost plot of Figure 5.2(b). Over the succeeding phases of learning the echoes evolve into highly structured trajectories and the Q-function takes on a quasi-monotonically increasing structure. In summary, Figure 5.2 illustrates the difficulty of static analysis of the echo states. The dynamics of the echoes evolve through time as the attractor underlying the reinforcement learning domain is discovered.

### 5.1.6 Finite Horizon Training

The most compelling reason for modeling a system via dynamic bases is that the basis may be trained, optimally, through linear regression [30]. Reinforcement learning, however, is an inherently on-line process. The optimal state-action value function defining the expected

[1]The MCP was learned and the three largest weights were selected from the final configuration of the readout.

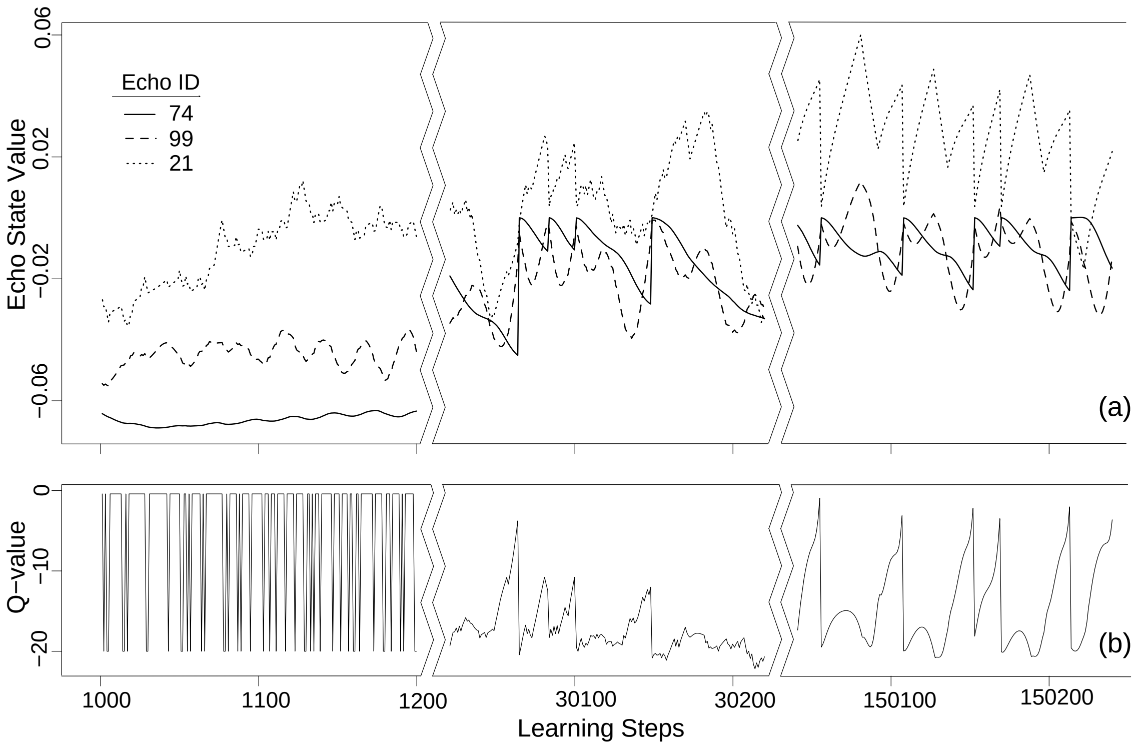

Figure 5.2: Example of echo state dynamics as the attractor structure changes through time in three parts: (a) initial training, intermediate attractor structure, and optimized attractor structure. Trajectories are plotted for the three most highly weighted echoes (echoes 74, 99, and 21, of 103 total, in decreasing order of weighting); (b) the corresponding approximate Q is depicted for reference.

sum of future rewards over an infinite horizon is never known, but the direction of this function with respect to the current approximation is known. This direction is given by the temporal difference error (TD), defined previously in Equation 2.3. Typically, when learning the state-action value function the TD-error is used as a gradient.

Gradient descent is not always necessary. The future can often be well-approximated with a finite horizon [22] as approximated in Equations 2.10. When a finite horizon is admitted in the training of the model, then a specific target, $Q_t$, can be constructed for the regression problem. Using this target, the ESN readout can be trained to project the maximal sum of future rewards over a finite horizon onto the ESN reservoir via linear regression, as defined in the training of the ESN, Equation 3.11. For the purposes of establishing the utility of the ESN in modeling the MCP reinforcement learning domain, a finite horizon was assumed, having length $h = 20$.

This horizon length, and those selected for experiments described in Chapter 6, were chosen based on theoretical and empirical results of past research [31, 32], as well as practical experience working with ESNs when modeling reinforcement learning domains. The maximal memory capacity of the ESN is the reservoir size, $N$ [31]. In practice, however, the effective memory length is much less, on the order of $N/2$. Experience shows that the maximal horizon length that may be accurately modeled via ESN is $N/2$ with performance increasing as this length decreases below this upper bound. This is a useful rule-of-thumb to use when modeling reinforcement learning with ESN. The horizon length should be chosen long enough to capture the entire length of the near-optimal trajectory. The ESN necessary to model this horizon should have a size $N > 2h$, with performance increasing as $N \gg 2h$. The exact size of the ESN also depends on the problem constraints. In this chapter the initial position may be anywhere on the range $x = [-1.2, 0.5]$. In the next chapter the initial position will always be a fixed state. When the range of initial conditions are smaller, the ESN may be of a size closer to the ideal outlined in the rule-of-thumb.

### 5.1.7 Attractor Mobility Effects on Batch Versus On-line Updates

One particular motivation for using linear regression to solve for the linear weights mapping the reservoir onto the target is the noise of the ESN's dynamic basis. As it is generated stochastically, the basis contains a distribution of dynamic features. It would be expected that at any given time in the learning process, a large fraction of the ESN's dynamics will not be useful in modeling the reinforcement learning domain's underlying attractor. In fact, it would be expected that only a small fraction of the echoes are relevant to the problem's dynamic structure. Therefore, while the ESN update equations are deterministic, Equations 3.9 and 3.10 the echoes contain structural noise that must be addressed.

In the past, the ESN was used in nonlinear system identification tasks in which the attractor is stationary. In these instances, batch and recursive linear regression for least squares error minimization are equal. In the case of a mobile attractor the model parameters and the resultant attractor interact via the system's dynamics. Therefore the batch and on-line update schemes differ drastically in building domain representations.

Various methods of state-action value update may be employed. For the purpose of this research the goal was to select a method that works for the ESN architecture so that the feature space, as opposed to the training method, may be examined. Unlike complete state representations, the ESN state-space reconstruction is a nonlinear projection of historical state information. The echo is context. Unfortunately, the echoes are not one-to-one mappings.

This is a crucial point of working with ESNs. The echo state at some time, $t$, depends on the pathway taken to achieve the system state at time, $t$. The historical pathway is embedded, implicitly, in the echo state. Embedding, however, can be harmful when random actions are selected to explore the state-action space. No clean way of dealing with this problem is apparent. When a batch update is used to train the ESN's readout, however, the interactions of the echoes, and inherent noise of the random moves embedded in the context, are lessened. For the MCP experiments I employ a windowed update rule where the window varies from $20\times10^3$–$50\times10^3$ steps, depending on the complexity of the problem.

## 5.2 Dynamics of ESN Learning of Mountain Car

Analysis of the attractor underlying the MCP in Chapter 4 demonstrated that, while fully defined in a single, two-dimensional state representation, the attractor also resides in four piece-wise one-dimensional spaces, as shown in Figure 4.10. These one-dimensional spaces are constructed with trajectories determined by the gradient information of the state-action value function and are partitioned by the fixed point structure of the reinforcement learning domain. These linear spaces are the intrinsic feature space of the problem.

A natural way to decompose the MCP's domain is, therefore, to consider the Q-values not as a function of position, velocity, and action as is the case of Markovian reinforcement learning domains, but as trajectories (i.e., recursive functions) in state-action space. The motivation behind utilizing a dynamic basis, such as ESN, to model the MCP reinforcement learning domain is that it naturally produces trajectories in this space. The echoes are nonlinear dynamic projections of the current learning trajectory. By working in trajectory space the ESN avoids the types of bifurcations experienced by Markovian function approximations. The ESN generates a set of linear trajectories which are then combined linearly to form the underlying attractor. Thus, it would be expected that minimal bifurcations occur during learning. Rather, it would be expected that the underlying attractor smoothly changes as the weightings of individual trajectories adapt through experience.

### 5.2.1 Fixed Point and Stability Computations

To test this assertion, fixed point analysis of the attractor underlying the MCP, modeled via ESN, was performed during learning. This analysis was analogous to that done in Chapter 4, but was changed to accommodate for the ESN lacking a velocity input. The fixed points of the ESN can be computed analogous to that of Markovian approximations. The stability, however, cannot be computed similarly, because the initial input cannot be perturbed $\pm\Delta\dot{x}$. Further, a problem with computing the fixed point structure is the influence of the echoes in control decisions. With the ESN's echo state reset to $\mathbf{0}$, the Q value, and therefore action, $a$, and velocity, $\dot{x}$, are completely determined by $x$, $b$, and the linear readout weights associated with these inputs. To compute the fixed point and stability the following technique was employed. After reset the system was iterated. As in Chapter 4, the fixed point location was computed where the velocity of this iteration was equal to zero. From this new position, positive and negative actions were taken, approximating the positive and negative velocity perturbations. The velocity between the the first and second iterates were computed to assess the stability of this point. Of course, the accuracy of this stability classification, computed in this manner, will be lower than a direct perturbation of velocity, but it is suitable for the purposes of fixed point analysis in a simple attractor.

### 5.2.2 Attractor Mobility with ESN

An example of fixed point and stability analysis for a single learning trial of ESN on MCP is depicted in the bifurcation and performance trace of Figure 5.3. As was predicted, attractor bifurcations, and even fixed point changes, are less frequent for the ESN as compared to the local and non-local Markovian function approximations studied in Chapter 4. After an initial learning period, approximately $35 \times 10^3$ steps, during which mobility events occur with some frequency, the fixed point structure settles. We see a major bifurcation event occurring at 105E3 training steps. A selection of attractor plots made in conjunction with the example mobility trace are shown in Figure 5.4, and are used to graphically depict the impact of this bifurcation event on escape performance, which is seen to take a large drop at this point in learning. Two attractor structures leading up to this bifurcation are depicted in Figures 5.4(a) and 5.4(b), generated at $95 \times 10^3$ and $100 \times 10^3$ training steps, respectively. In these plots the unstable spiral rotation tightens. A possible explanation for this is that the ESN is trying to minimize path length. A tighter spiral would shorten the path from points in state space that have the longest minimum path to the goal. The result, however, is a limit cycle formation at $105 \times 10^3$ learning steps. The attractor structure returns to a loose unstable spiral at $110 \times 10^3$ learning steps.

The limit cycle formation at $105 \times 10^3$ learning steps is the cause of the precipitous drop in escape efficiency observed. Trajectories initialized at $\mathbf{s} = \{-1.2, 0\}$ are captured by the limit cycle. Because of the inelastic nonlinear boundary at $x = -1.2$, nearly two-thirds of the trajectory state space is captured by this cycle, making escape without restart nearly impossible within a policy having minimal random action.

This example highlights a drawback of using dynamic features in reinforcement learning. Dynamic features have, potentially, global impact on the underlying attractor of the reinforcement learning domain. Conversely, the early phases of learning, characterized by high exploration probability have less impact on the feature's structure. Trajectory based features are amalgamations of many action decisions and are difficult to perturb.

This result, in conjunction with the mobility traces and attractors of Chapter 4 suggest that the feature representation itself, rather than exploration, determine the mobility of the underlying attractor, and that the ESN initialized with a robust, global representation of the problem domain. In succeeding sections ESN escape performance compared to traditional architectures in the Markovian domain in both a static and dynamic context will be discussed followed by an analysis of how the ESN is approximating this state space and an assessment of the this approach's strengths and weaknesses.

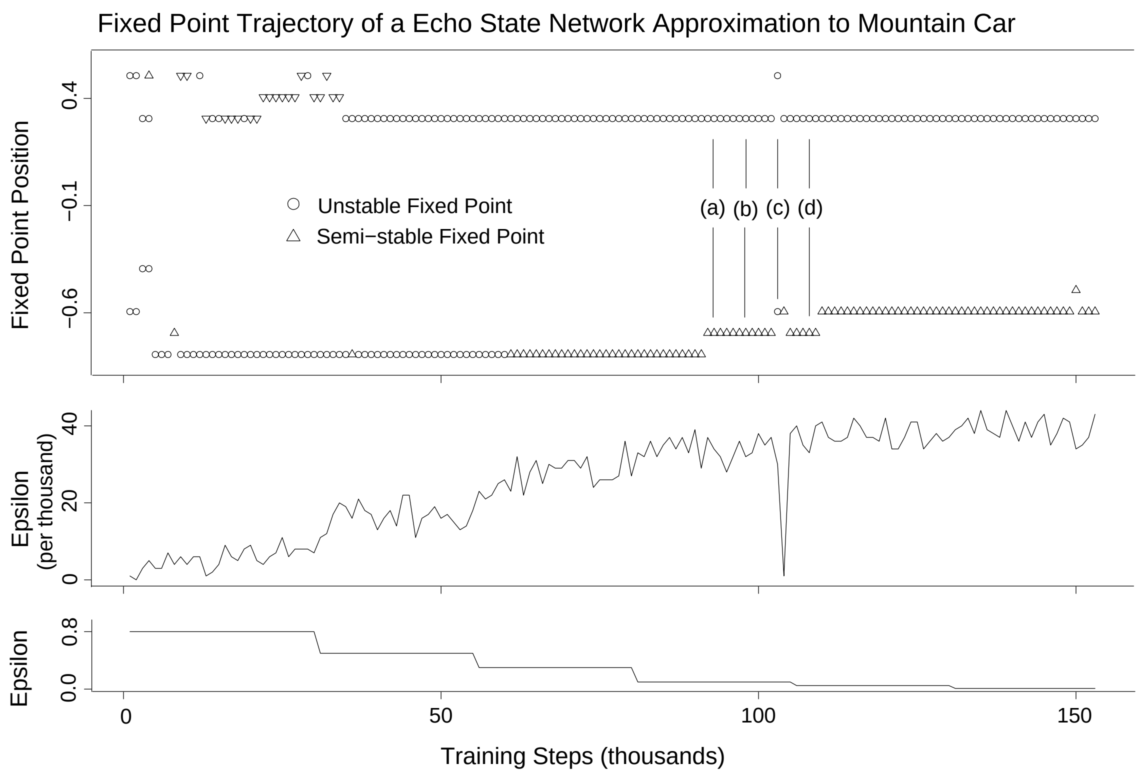

Figure 5.3: Bifurcation diagram of an echo state network approximation to the Mountain Car Problem during learning. Fixed point based representation of attractor structure is highly global. A global bifurcation event is visible in the fixed point structure at $95 \times 10^3$, $100 \times 10^3$, $105 \times 10^3$, and 110E3 learning steps. This event is explored as trajectories in state-space in Figure 5.4, corresponding to points (a–d).

## 5.3 Performance Evaluation

MCP is a pathway minimization problem. Escaping the valley, itself, is not a challenging problem. The attractor structure is simple enough to be satisfied by suboptimal approximations. Rather, the performance of an architecture on this problem is measured in three ways: decision quality (measured as minimal decisions to achieve goal), model complexity (number of trainable parameters), and training complexity (number and computational complexity of training steps). In the case of the latter two criteria, they are measured with respect to an achieved level of decision quality. Stemming from the lack of comparable architectures for modeling continuous valued, non-Markovian reinforcement learning domains, the performance of the ESN is compared to traditional mechanisms for solving these domains: a lookup table approximation and neural network function approximation. The

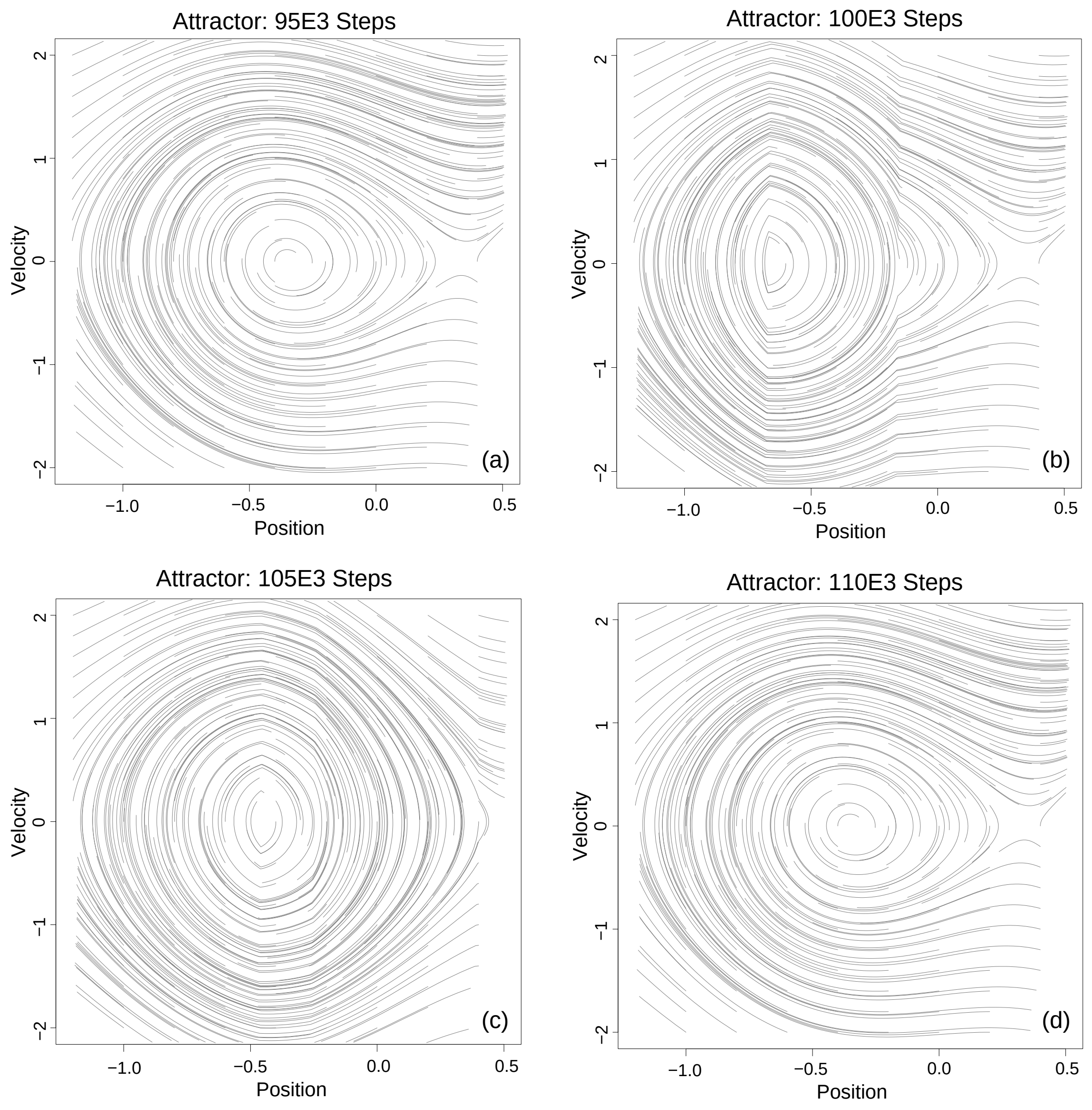

Figure 5.4: Example MCP attractors constructed from learning interactions between the system dynamics and the echo state network function approximation. These plots correspond to fixed point structural events presented in Figure 5.3.

lookup table serves as a performance upper bound. The neural networks provide realistic performance assessments—numbers of trainable parameters compared to input information (complete versus incomplete state space) to gauge the level of performance that the echo state feature space can achieve.

The direct performance comparison was constructed as follows. 150 learning trials were performed using the ESN construction parameterized (as described above) for each of two training methods: backpropagation of TD-errors (learning rate, $r = 0.0005$, discount factor $\gamma = 0.9$), and linear regression over a finite horizon of length $h = 20$. The ESN received an incomplete input state vector $\mathbf{s} = \{x, a\}$.

To compare the performance of local Markovian, non-local Markovian, and non-local non-Markovian approximations, the following experimental set-up was used. Each architecture was trained on the MCP, parameterized as described in Chapter 4. 150 learning trials were performed for lookup table and neural network architectures defined in Chapter 4. A smaller (25 hidden unit) neural network architecture was also included. All Markovian architectures received the complete MCP state, $\mathbf{s} = \{x, \dot{x}, a\}$.

### 5.3.1 Results

The results of the performance comparison are summarized in Figure 5.5. Figure 5.5 is a histogram in which performance is measured as the maximum escapes per thousand training steps (EPT) observed over the trial. This is recorded on the x-axis. The percentage of trials in which this performance was observed is recorded on the y-axis. Performance is discretized such that 25 bins span the performance envelope of all architectures.

Results of this experiment are not surprising. The best performing architecture is the lookup table, which converged to a near-optimal solution in 100% of all trials of approximately 56.5 EPT. This result outperforms all non-local approximation architectures studied. This performance profile can be considered the best possible result for the purposes of comparison.

The remaining four plots depict results of the non-localized function approximations. Assuming normal distribution of the data, the 200 parameter ESN trained via linear regression, $ESN_{lr}$, performance difference was statistically significant compared to the 100 parameter neural network, $NN_{100}$, with significance ($Pr < 8.854E-8$) and the 200 parameter neural network, $NN_{200}$, with significance ($Pr < 0.001952$). In this comparison, $NN_{200}$ performed best with $36.87 \pm 1.91$ EPT. This outperformed $ESN_{lr}$ having a performance distribution of $34.86 \pm 7.22$ EPT and the performance of $NN_{100}$ which achieved $31.69 \pm 1.19$ EPT. These performances were significantly better than that of ESN trained via backpropagation of TD-errors, $ESN_{td}$, which achieved only $11.74 \pm 2.75$ EPT.

Intuition with respect to the plot of $ESN_{lr}$ performance in Figure 5.5 suggests that that $ESN_{lr}$ performance is multimodal, rather than normally distributed, with one optima centered around approximately 40 EPT, and at least one other optima correlated to the distribution of $NN_{100}$ and possibly a third mode correlating with the performance of $ESN_{td}$, respectively. This multimodality, however, is more likely an artifact of locally optimal control strategies that achieve sub-optimal MCP performance. This result makes sense because the performance of the ESN architecture is known to distribute normally for systems in which the attractor is static. If a mobile attractor is considered to be a sequence of static attractors, then the performance of the ESN would be expected to exhibit multimodality. Reservoirs having poor features would fail to capture portions of the underlying attractor. Once the probability of exploration dropped to zero, these ESNs would converge to suboptimal policies.

## 5.4 ESN Feature Analysis

A challenging aspect of working with dynamic bases is determining how these architectures reconstruct the state-space. The ESN's echo states are high-dimensional, random dynamic projections of the underlying system dynamics, but to be successful they must exhibit properties that would be expected of a Markovian representation of the system. Markovian states connect locally in physical (i.e., real-world) domains. Therefore, I would also expect the echo state representation to exhibit local connectivity.

I can assess, qualitatively, the local connectivity of the echo space by clustering the echo states via K-means. Using the learned cluster centers as labels, I can assign an echo state

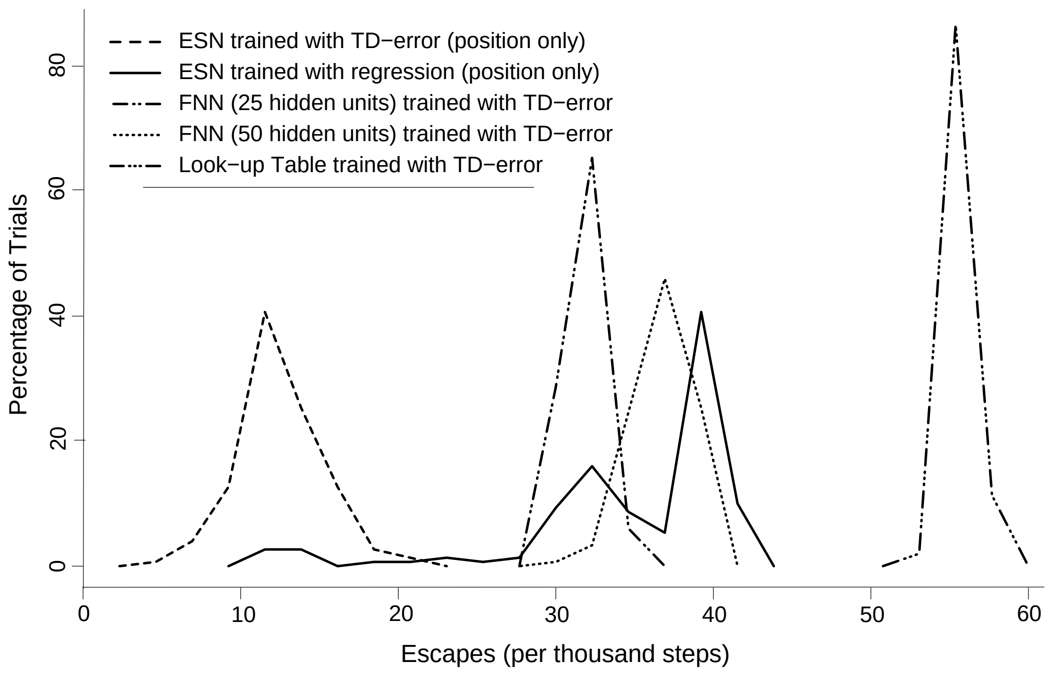

Figure 5.5: MCP performance comparison of Markovian and non-Markovian approximation critics: ESN trained via TD-error, ESN trained via regression of a finite horizon, $h = 20$, FNN (25 hidden units) trained via TD-error, FNN (50 hidden units) trained via TD-error, and a lookup table, discretized at increments of 0.1 on the state space.

cluster label to each point in the complete state-space of the MCP. The labels act as a gross visualization of the Markovian reconstruction of the non-Markovian state-space. This visualization is presented in Figure 5.6(a). In this plot, individual symbols represent the labels of the clusters, $K = 20$.

A few details of this representation should be discussed. First, the observable incomplete state of the problem was position, $\mathbf{s} = x$. Therefore, the echo states must reconstruct a velocity representation and map this representation onto the position. Thus, the echo state space should cluster both on the $x$-axis, and, where the velocity plays a significant roll in the expected sum of future rewards, on the $\dot{x}$-axis as well.

In fact, it is observed that the echo states cluster in a spiral pattern, which falls roughly along the lines of the partition of the underlying piecewise linear space depicted in Figure 4.10. This clustering pattern is a fundamental decomposition of the domain, differentiating between points of similar position only when this differentiation is important in determining the expected sum future reward signals.

The approximate minimum path from the most negative Q-value to the goal has been superimposed onto Figures 5.6(a) and (b) for reference. This line allows the inefficiency incurred by the echo state-space, as marked within the gray boxes, to be easily observed. Echo state clusters cross over the minimum path line. This is a suboptimal representation because points below this line, optimally, map onto $a = -1$ and points above map to $a = 1$. This line marks the policy decision boundary and ensures the minimum path to goal is preserved.

Because the echo states are not properly distinguishing between regions of the complete state-space with significantly different trajectories in the optimal MCP attractor, we know that the ESN is operating sub-optimally. Why is this the case? The answer requires thinking in terms of how the ESN represents the state space—dynamic projections of the system trajectory. Trajectories both above and below the policy decision boundary are similar. Only when trajectories near the decision boundary approach the saddle point at $x \approx 0.4$ does the differentiation become clear. What the ESN lacks is a global awareness between trajectories above and below the line, which in feature space reside very far apart, as depicted in Chapter 4, Figure 4.10. This issue will addressed in detail in Chapter 7, where contextual insufficiency of echoes can be addressed.

Figure 5.6(b) provides a global perspective of the echo state-space's discrimination power. In this figure the actual policy selected by the agent is plotted at each state visited. In MCP the optimal policy rule, with few exceptions, is to maximize acceleration. The policy should include as few no-operations ($a = 0$) as possible. As is evident in Figure 5.6(b), the ESN solution has few no-operations. Figure 5.6(b) also confirms the concerns addressed previously—the representation problems made along the policy decision boundary produce sub-optimal actions. Actions below the policy decision boundary on the range $x \approx [-0.5, 0.3]$ and above zero velocity should be in the negative ($a = -1$). In this plot, the pathway is sub-optimally kept long by pushing the car as far up the positive valued valley as physics will allow before retouring the valley floor.

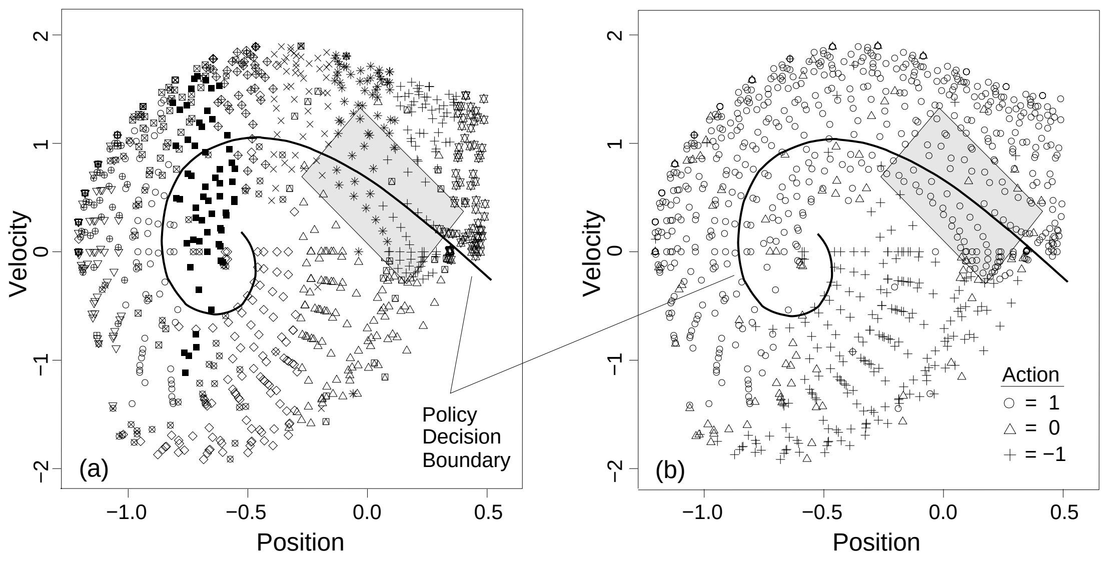

Figure 5.6: Analysis of echo state reconstruction of the Markovian state-space. (a) Markovian state-space visited during a segment of learning in the MCP for the ESN critic. Point labels correspond to the cluster ids of the K-means clusters of the echo states used to predict $\mathbf{Q}$ of this state. (b) Corresponding policy decisions chosen by the ESN critic.

## 5.5 Temporal Learning Comparisons

Previously in this chapter the performance of the ESN architecture was compared to that of local and non-local Markovian function approximations statically. One must also consider the temporal evolution of the performance of these architectures to understand, not just what is learned, but how this performance evolves through time under training schedules of varying lengths. This analysis is composed of two parts, a performance component, in which the time-evolution of escape efficiency is compared among architectures are compared, and a dynamic analysis component in which the time-evolution of the attractor is studied alongside performance.

### 5.5.1 Performance Comparisons

Temporal evolution of learning efficiency on MCP was studied by varying the length of the learning schedule and the architecture used to approximate the state-action value function, $Q$. Learning schedules were used to incorporate random actions into the policies of the non-local function approximations. For the neural network architectures, the learning schedule was $S_{nn} = [0.8, 0.8, 0.5, 0.1, 0.01, 0.01, 0.0]$. The ESN architecture was found to train best on a slightly different schedule $S_{esn} = [0.8, 0.8, 0.5, 0.3, 0.1, 0.05, 0.01]$. For each level of randomness the agent was permitted to learn over a finite length of steps such that $step = [1000, 5000, 5000, 5000, 5000, 5000, 5000, 5000]$. Further, these finite intervals of learning were repeated $N$ times, such that the number of training steps equaled $[[1000_1, 1000_2, ..., 1000_N], [5000_1, 5000_2, ..., 5000_N], ...]$. These schedules enabled precise control over the rate at which the randomness injected into the policy dissipated compared to number of experiences observed. To examine performance of the architectures over a range of learning speeds, the number of iterations over each element of the learning schedule, $S$, was varied, $N = 1, 2, ..., 5$. For each architecture, and for each value $N$, thirty learning trials were run. The number of escapes per thousand training steps (EPT) was recorded at intervals of 1000 steps. The final five EPT values were averaged for each trial and this mean was averaged over the thirty learning trials. The results of these experiments are summarized in Table 5.2. For each configuration the mean of the final five maximum results over all the thirty trials was also recorded. These results are summarized in Table 5.1.

Table 5.2 contains two results of interest. First, the mean escape efficiency of the ESN under learning schedules of length $93 \times 10^3$–$155 \times 10^3$ are comparable to that of the non-local function approximations. The local lookup table results are presented for means of comparing the ESN results to that of a solution that converges to a near-optimal policy for the MCP. The second result is that for the shortest length schedules, $31 \times 10^3$ and $62 \times 10^3$, the ESN performs significantly worse. This is surprising. One possible explanation is the length of the linear regression window, which was fixed at $50 \times 10^3$ steps for these experiments. For the shortest length schedules, a window of this size would include a large percentage of high exploration echo states. While random action states are not included in training, these actions are implicitly embedded in the echo states which follow. This could add significant noise and negatively effect precision of the linear readout.

One concern of the ESN architecture so far overlooked is the stochasticity of its features. ESNs are commonly compared to alternate techniques by training multiple ESNs and then comparing both the distribution and the best possible result against architectures having

| Maximum Escapes (per thousand steps) | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | Learning Schedule Length (steps) | | | | |
| Input | Model | Weights | 31E3 | 62E3 | 93E3 | 124E3 | 155E3 |
| | *Table*[†] | *2970* | *23.5* | *40.0* | *49.0* | *54.0* | *56.5* |
| $[\mathbf{x}, \dot{\mathbf{x}}, \mathbf{a}]$ | $NN_{25}$ | 126 | 36.7 | 37.4 | 36.2 | 36.0 | 35.8 |
| | $NN_{50}$ | 256 | 35.9 | 33.5 | 35.1 | 35.5 | 36.1 |
| $[\mathbf{x}, \mathbf{a}]$ | $ESN_{lr}$ | 206 | **39.4** | **39.8** | **43.4** | **42.6** | **42.6** |

Table 5.1: Temporal comparison of maximum escape performance for varying length learning schedules. [†]Note, no learning schedule was used in conjunction with the lookup table approximation, only an upper limit on training steps was employed.

learned features. This is true for stationary attractors and should hold for nonstationary attractors as well. On average the ESN performs about as well as trained Markovian architectures at longer schedule lengths but less well on higher-speed training schedules. However, if we observe the maximum result over all 30 trials, Table 5.1, the ESN is the superior nonlocal function approximation method. Therefore, over a set of learning trials, the distributions defining the reservoir generate a reservoir instance that exhibits high-performance. This result held over all five training schedule lengths when compared to the neural network function approximations.

This result is beneficial in understanding the strengths of the dynamic basis model of reinforcement learning. For any given trial, we expect performance to be on average about as good as that of an adapted representation. However, adaptation can induce local optima. In this case, allows the global optima for the current instance to be realized. Thus, if we generate enough instances of the ESN, our expectation for identifying near-optimal solutions increases.

### 5.5.2 Mobility Comparisons

The final analysis of dynamic representations of the MCP examines how each architecture studied interacts with the problem. To create a global perspective of attractor structure the following experiment was performed. For all trials used to create the performance experiments in the previous section, the fixed point structure of the attractor during learning was recorded. A *mobility event* was defined as a change in the fixed point structure between

| Mean ± Std. Dev. Escapes (per thousand steps) | | | | | |
|---|---|---|---|---|---|
| | Learning Schedule Length (steps) | | | | |
| Model | 31E3 | 62E3 | 93E3 | 124E3 | 155E3 |
| $Table^{\dagger}$ | *15.3±2.6* | *26.9±3.7* | *41.7±3.6* | *45.7±3.6* | *49.9±3.1* |
| $NN_{25}$ | 22.6±15.2 | **28.6± 8.6** | 28.5± 5.0 | 28.1± 8.4 | 27.9± 8.2 |
| $NN_{50}$ | **27.2±11.0** | 27.3± 6.0 | 28.9±3.9 | **29.6± 3.4** | **30.1± 3.2** |
| $ESN_{lr}$ | 17.8±11.5 | 19.4±12.8 | **29.4±10.8** | 29.2±11.6 | 29.0±12.3 |

Table 5.2: Temporal comparison of mean escape performance for varying length learning schedules. *Note, no learning schedule was used in conjunction with the lookup table approximation, only an upper limit on training steps was employed.

successive observations at a discrete position of the state-space, discretized $x = [-1.2, 0.5]$ at intervals of $\Delta x = 0.1$. Observations were made every time performance was recorded, every 1000 learning steps. For each time step, the number of mobility events were averaged, forming a single time-series for each trial. These time-series of mean mobility events were plotted with respect to the percent of learning schedule completed. Thus, all trials from the previous experiment can be compared, regardless of actual length of schedule. These results are summarized in Figure 5.7.

This picture, when taken in context with the performance results presented above and the attractor snapshots, provide enormous insight into how different architectures interact with reinforcement learning domains. The lookup table representation exhibits a high number of mobility events throughout the early stages of learning, despite substantial performance increases as presented in Table 5.2. At time periods equivalent to the end of the learning schedule, the table's mean escape efficiency approaches 88% of the optimum, while exhibiting a little more than two mobility events. The attractor is still undergoing structural changes, but these changes are very small. The mobility events must be taken in context of the representation scale. The size of the feature determines the impact of the mobility event on the global attractor structure.

In contrast, the ESN architecture maintains a relatively low number of mobility events throughout learning, even under very high exploration phases of the learning schedule–under one event per thousand in the early learning stages to approximately 0.25 mobility events per thousand under the later low-exploration phases of learning. Despite the lack of fundamental

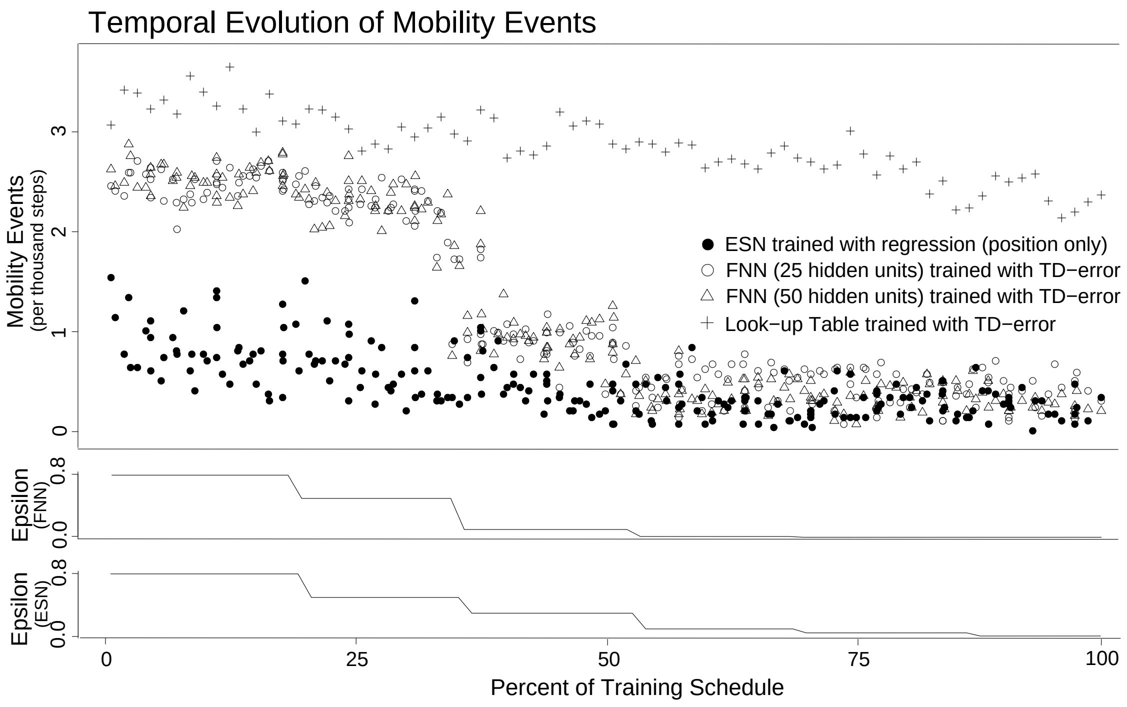

Figure 5.7: Temporal evolution of mobility events across all critic architectures and all training schedule lengths. Mobility events were plotted where temporal information was mapped to percentage of training schedule completed.

changes in the attractor structure, Table 5.2, shows marked performance improvement over this learning curve. Moreover we see a wide variance in learning over the entire learning schedule. From this we can expect that, while the attractor fixed points are not changing, the nonlinear structure around these fixed points is changing. Circumstantial evidence, depicted in Figure 5.4, supports this view of the ESN's low mobility profile. The structure of the unstable spiral is changing, even as the fixed point at its center remains stable. ESN mobility events also seem to correlate with global shifts in the attractor structure, unlike the highly localized events exhibited by the lookup table.

Most interesting are the changes in the attractor structure induced by the nonlocalized, Markovian feature representations learned via the neural network architecture. The attractor structure seems to exhibit properties of both localized and nonlocalized feature representations.

In the early phases of learning, as depicted in the types of attractor structural changes shown in Figure 4.9, the mobility effects are both numerous and highly nonlocal. The attractor makes large iterative changes in structure and is very susceptible to exploration. However as exploration decreases the attractor structure quickly settles and the number of mobility events more than halves. This is also supported circumstantially by Figure 4.9 in which it can be observed that the attractor structure late in learning is focused on non-linear deformation of the unstable spiral as opposed to changing the fundamental dynamic structure, which has a fixed point effect of shifting the location rather than the stability class of the fixed point.

## 5.6 Discussion

What insight can be drawn from applying the ESN architecture to solving the MCP? First, the ESN is quite capable of building high-quality representations of this domain. The ESN is competitive with traditional Markovian nonlocal function approximation structures despite receiving only non-Markovian input. The ESN solves the problem in a unique way, relying on linear combinations of trajectories to determine the current control decision. This allows the ESN to work in the intrinsic feature space of the problem—the dynamic structure, which exhibits lower dimensionality. This implies that the ESN should scale to higher dimensional problems more easily than state-space representations. This will be explored more in Chapter 6.

There are, however, inherent drawbacks of using the ESN's dynamic representation. First, on a path minimization problem, the negative effects of transients and the implicit embedding of random actions in the echo state likely induce decision inefficiency. Second, it is difficult to predict the dynamics of a reinforcement learning problem *a priori*. As was discussed in this chapter, the ESN has difficulty discriminating between local trajectories with long-term future consequences. This is a representation weakness of the ESN. If the ESN does not initially exhibit sufficient representation power, it will not adapt during learning. Particularly, no advantage was taken of the $Q$-values predicted by the ESN. These values are significant indicators of long term future dependencies. Therefore, the ESN was able to successfully predict Q for most non-Markovian states, but it experienced difficulty in discriminating between certain non-Markovian states where the local trajectories were similar but modeled vastly different Q-values. If predicted Q-value were used to embellish the echo states, the ESN could achieve a higher-level of representation, making the non-Markovian structure more robust to long-term temporal dependencies. A method for doing this has already been developed and its use in the reinforcement domain will be discussed in Chapter 7.

# Chapter 6

# Modeling Complex Decision Problems with ESN

In past chapters the dynamics of a simple non-Markovian reinforcement learning domain, the MCP, were investigated. While a good benchmark for study and comparison of the ESN architecture to past, complete state representations, other domains are required to ascertain the scalability of the ESN architecture; scalability in terms of either greater temporal complexity or greater dimensionality. To identify the scalability of ESN two non-Markovian reinforcement learning case studies are presented: Modified Mountain Car and Acrobot. Dynamically, these problems are analogous to the single and double pendulum swing-up problems, respectively, differing primarily in the parameterization of the state-space and constraints on the action space. These domains provide a measured scale-up from MCP. In Modified Mountain Car, the attractor complexity increases over MCP in a predictable way, allowing for comparison to the studies in Chapter 4 and 5. Acrobot is the capstone case-study for this research. This domain exhibits complex dynamics, higher dimensionality, and multiplicity of goal states that does not occur in either of the Mountain Car variants. Successful learning of this problem in non-Markovian state-space via ESN marks an incremental improvement in reinforcement learning.

## 6.1 The Modified Mountain Car Problem

The Modified Mountain Car Problem (MMCP) is a more complex and challenging variant of the Mountain Car Problem. The system dynamics of the problem are identical, but the

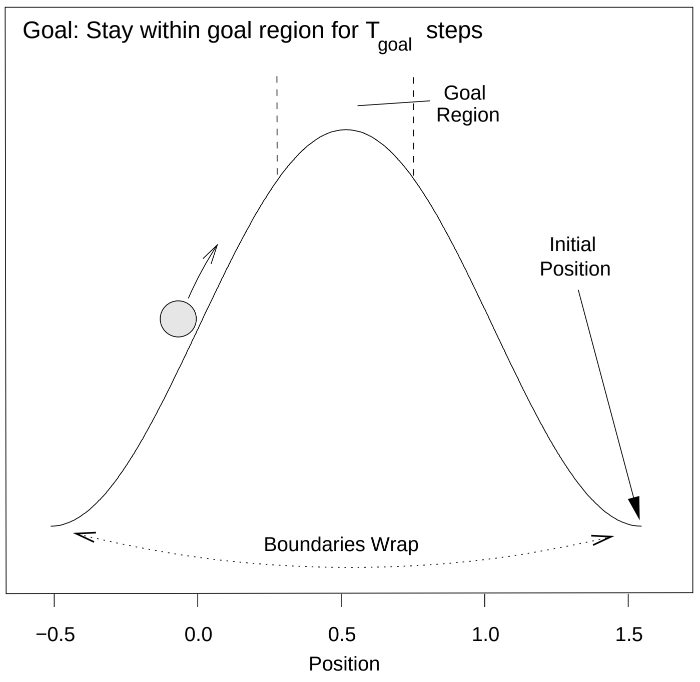

Figure 6.1: Diagram of the Modified Mountain Car Problem

.

problem constraints and the learning objective are different, which induce changes to the attractor underlying the learning problem. In the MMCP, depicted graphically in Figure 6.1, the perfectly inelastic wall and the goal state are removed. The boundaries of the state-space are shifted such that $x = [-0.524, 1.572]$. Shifting the bounds of the problem centers the peak of the mountain in the center of the state-space at $x = 0.523$. Moreover the left and right boundaries are defined such that they bisect the valleys on either side of the peak and the endpoints of the state-space wrap (i.e., when the position goes beyond the state-space range the agent proceeds with equivalent velocity from the opposing boundary. The goal-state of the problem is also changed such that the goal is achieved if $0.373 < x_g < 0.673$. For additional complexity, the goal-state is parameterized such that $x_g$ must be achieved during $T_{goal}$ contiguous time-steps. This constraint requires the agent to balance the car within $\Delta x$ of the peak of the mountain. Modifying the MCP in this way effectively transforms the problem into a variant of the pendulum swing-up learning problem with the requirement that the pendulum remain balanced for at least $T_{goal}$ steps.

### 6.1.1 Dynamics of the Modified Mountain Car Problem

The MMCP, when learned, exhibits two dynamic regimes, shown in Figure 6.3. Initially, the problem is identical to the MCP. Without *a priori* knowledge, a *no-operation* or *random* policy results in a global, stable attractor, as was seen in Figures 4.3(a) and (b). The attractor evolves in parallel to the attractor of the MCP until the agent achieves the goal state. Achieving the goal state facilitates the formation of a small, stable limit cycle circumscribing the peak of the mountain. This limit cycle's radius is determined by two factors, either the allowable limits of the balanced region, $\Delta x$, or, if $\Delta x$ is sufficiently large, the physics of the system. As was the case of the MCP, the MMCP attractor exhibits saddle points between the mountain peak at $x = 0.523$ and the valley floor at $x = 1.572$ and $x = -0.524$, respectively. Because the MMCP is symmetric, two such saddle points emerge that define the intersection of the unstable spiral centered on the valley floor and the limit cycle centered about the mountain peak. These structures and fixed points are highlighted in Figure 6.3(a).

The attractor structure of MMCP is incrementally more complex than that of the MCP by the addition of the stable limit cycle. This incremental change, in the dynamic sense, however, is a much more challenging problem. The model must be able to represent two dynamic structures, which exist at two different levels of granularity. The large unstable spiral requires a gross policy with long sequences of opposing actions to move the car out of the valley. In contrast, the goal region requires a fine-grain policy to balance the car within the goal region for $T_{goal}$ training steps to achieve the goal state.

### 6.1.2 Reward Shaping

The complexity of the MMCP raises an additional problem beyond that of attractor representation. Unlike a lookup table approximation, which, through fine-grain storage of past experiences, guarantees the goal will eventually be achieved, function approximations provide no such guarantees. As described in Chapter 4, a combination of random exploration and deterministic exploitation of the learned Q-system is used; first to achieve the goal and then incrementally refine the minimum pathway from the initial state to the goal state. In difficult learning problems, however, the ease with which a learning schedule may be constructed such that function approximation can initially achieve the goal state diminishes. This is a well-known problem and has been studied extensively [54]. The solution to this problem is a technique called *reward shaping*. The premise of reward shaping is a carrot-and-stick approach. Initially, the goal is defined as an easily achieved constraint compared with the final, desired goal state. Intermediate goal states are used to slowly transition from an initial "easy" goal, to the final "challenging" goal. Reward shaping enables the agent to learn intermediate Q-systems while ensuring the agent achieves goals regularly. Regular experience of the goal state is critical to function approximation because of the tendency for approximations to forget past experiences [48, 64, 21].

In the experiments of this chapter, reward shaping was used to transition the problem from that of a swing up problem, identical to the MCP, to a swing up and balance problem, the MMCP. As described above, the goal state for the MMCP is defined spatio-temporally. The agent must position the car in the goal range $0.373 < x < 0.673$ for $T_{goal}$ consecutive steps. Initially the value of the $T_{goal}$ is relatively small. This goal allows the agent to quickly learn the swing-up problem. Slowly, over many learning trials, the value of $T_{goal}$ is raised such that the agent transitions from learning swing-up only to swing-up and balance, which has a more complex policy.

### 6.1.3 ESN Learning Performance

The following experiments were run to test ESN modeling performance on the MMCP. The agent always observed the same initial state, $s_{init} = (x_{init}, \dot{x}_{init})$, where $x_{init} = 1.572$ and $\dot{x}_{init} = 0$. This corresponds to the car starting at the base of the valley with no initial velocity. Initially, the goal was defined as balancing the car within the goal region for $T_{goal} = 3$ time-steps (0.15 simulated seconds). The goal was shaped according to the following schedule, $T_{goal} = \{3, 5, 6, 7, 8, 9, 10\}$, where changes to $T_{goal}$ were invoked at time-steps $[20 \times 10^3, 35 \times 10^3, 50 \times 10^3, 80 \times 10^3, 110 \times 10^3, 140 \times 10^3]$. The final $T_{goal} = 10$ was defined to simulate a temporary balancing of the car for 0.5 simulation seconds, which was deemed appropriate to demonstrate the ESN had captured the fine-grain control of the car.

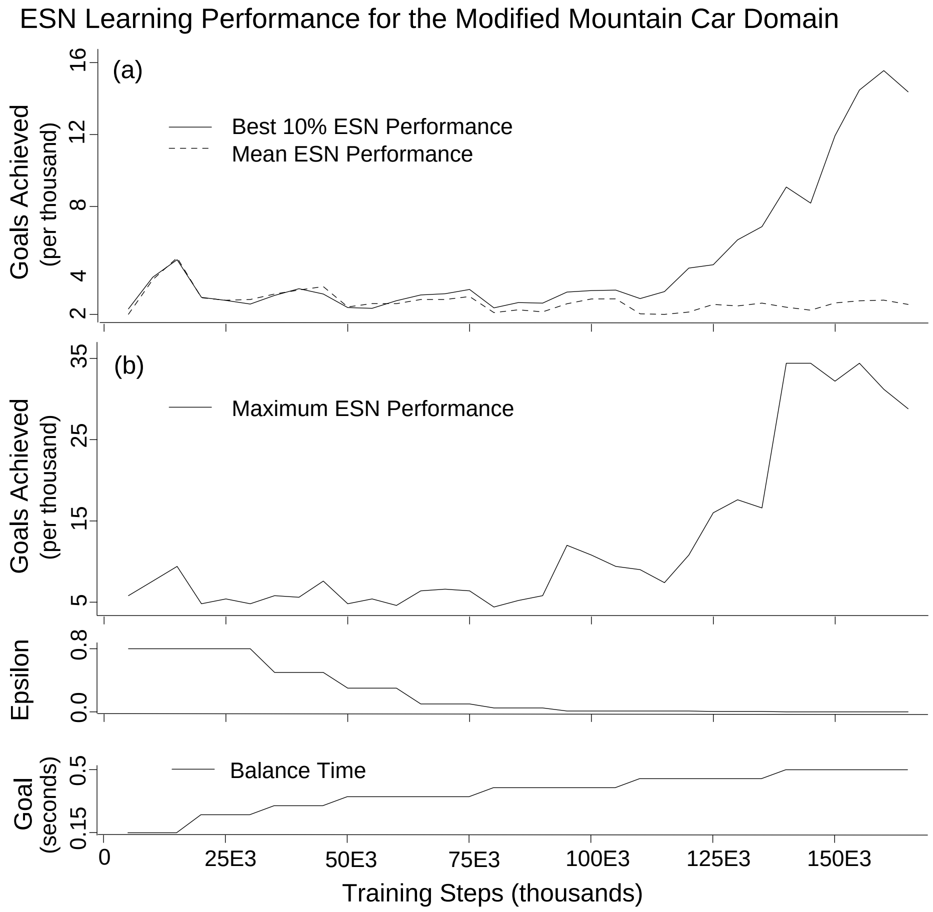

Figure 6.2: ESN modeling performance on the Modified Mountain Car Problem: a) overall mean and best 10% mean ESN performance; b) maximum ESN performance.

Indefinite balance is best achieved by linearizing the problem domain near the goal and utilizing two controllers, a learned nonlinear controller (such as the ESN) for initial swing-up and balance and linear controller for indefinite balance. By balancing for 0.5 seconds, the ESN is able to reduce the size of the non-Markovian state-space to a degree where a lookup table approximation to the non-Markovian domain is feasible by explicit temporal embedding.

The reward signal for the goal state was 0. All other states received a reward signal of -1. As in previous experiments, a learning schedule was used to balance exploration of the state-space and exploitation of the Q-value function approximation. The learning schedule was $[0.8, 0.5, 0.3, 0.1, 0.05, 0.01, 0.0]$. Changes between subsequent values of the schedule were invoked at time-steps $[35 \times 10^3, 50 \times 10^3, 65 \times 10^3, 80 \times 10^3, 95 \times 10^3, 110 \times 10^3, 125 \times 10^3]$.

Preliminary experiments indicated that the reservoir dynamics described in Chapter 5 were sufficient to represent the problem when the ESN size was increased to $N = 200$, twice the reservoir size required to represent the original MCP. As was done for the MCP, the embellished echo state, $z$, was concatenated with its square to better facilitate the asymmetry of the echo states. Thus, the total number of trainable parameters in this architecture was $2 \times 203 = 406$. The finite horizon for this problem was 150 training steps. As in the MCP, regression was used to train the linear readouts over the finite horizon of computed Q-values. Regression was performed every 1000 learning steps over a regression window of 20E3 historical training steps. 150 ESNs of this size were trained with these parameters. The results of this training are summarized in Figure 6.2.

Figure 6.2 is divided into two results. Figure 6.2(a) reports both the mean ESN performance over all trials, and the mean performance over the best 15 trials (i.e., best 10%). Note, escapes are averaged over each 5000 training step segment of the learning trials. Figure 6.2 shows that the average performance of the ESN was approximately 2 escapes per thousand steps. The best 10% of ESNs however, were able to satisfy the learning problem with mean policy lengths of 62.5 steps, clearly suboptimal, more than twice the estimated optimal path length of 27.9 steps.[1]

Figure 6.2(b) depicts the performance of the overall best ESN trial. This ESN achieved peak performance of more than 35 goals per thousand training steps, equivalent to an average of 28.6 steps per goal, very near the best performance expected for this learning problem.[2] These results indicate that the MMCP problem is significantly more difficult for the ESN to represent compared with MCP, even though the attractor structure of MCP is replaced by a single trajectory (i.e., the MMCP only has a single start state instead of a continuum of start states as described for MCP in Chapter 4). The reasons for this difficulty

[1]Near-optimal performance was defined as the maximum lookup table policy length of the MCP problem plus 10 additional balance steps.

[2]Presenting the data in terms of both *goals per thousand training steps* and *steps per goal* serves two purposes. The MCP data presented in Chapters 4 and 5 use the first set of units, which I consider the best indicator of performance over an attractor. The second set of units is appropriate for the MMCP and Acrobot Problem because all trajectories begin at a unique initial state.

are addressed below.

### 6.1.4 ESN Performance Analysis

The ESN performance distribution is generated by random initializations of the reservoir. Not all ESNs perform equally well, and therefore it is necessary to consider the distribution of ESN performance over many restarts to estimate its performance capabilities. While the best ESNs of the experimental trials have been shown capable of learning the MMCP, the severity of performance drop-off between the very best members of the distribution and the mean performance should be explained. What makes the MMCP so challenging for the ESN?

The answer is that the policy required to achieve the MMCP goal state is more complex and more difficult to represent. The attractor of the learned MMCP problem contains a second dynamic component beyond that of MCP, a limit cycle surrounding the goal region in the low-velocity region, which must be accurately represented to achieve the goal state. The likelihood of a given ESN reservoir containing sufficient richness to represent an attractor decreases as the level of complexity of the attractor increases [13]. Fewer ESNs will be randomly instantiated from a parameter set that have suitable richness. As is often the case in function approximation, finding the right parameters for the reservoir is paramount. Those reservoirs that do contain suitable internal dynamics, however, are capable of learning the problem, as demonstrated in Figure 6.2(b).

To demonstrate the complexity of the MMCP policy and to demonstrate how representation capability of ESNs on the MMCP distributes through different learning trials, two representations of the experimental data have been constructed, Figures 6.4 and 6.3(b), respectively.

Figure 6.4 depicts the unique policies learned for the best seven ESNs in terms of mean performance over the final 15E3 training steps, which correspond to full exploitation of the learned attractor, $\epsilon = 0.0$. Because all trials start at the same state, $x_{init}$, the policies may be presented visually, side-by-side, as tree traversals. Each node of the tree corresponds to a time-step of the system, and the edges of the trees represent actions. The depth of the tree corresponds to time. The shorter the traversal, the better the policy. Policies depicted in Figure 6.4 have been divided (depicted as different line types) into three classes: best policy found, equal to 28 steps; polices within twice the best policy length, and those policies greater than twice the best policy length.

Figure 6.4 facilitates visual recognition of several features of the data. First, the best policies are not, in fact, all contained within the best ESN. In fact, these policies are distributed throughout the best ESNs. Therefore, the performance of the ESN was not dictated by the number of best policies found, but by the number of times this policy was exhibited during this interval of the training schedule. It is also apparent that the best policy exhibits two structures. In the 1st, 4th, and 5th ESNs, the best policy exhibits long, steadily increasing sequences of contiguous -1 and 1 actions, which then terminate with small -1 and 1 action perturbations. This would be expected as the canonical policy of the MMCP problem. Contiguous action sequences constitute the policy of swinging out of the valley floor, and the final sequence of actions constitute the fine-grain, balancing decisions necessary to stay within the goal region long enough to achieve the goal state. The best policies exhibited by the 2nd and 7th ESNs, however, are quite different. These policies exhibit large contiguous sequences of actions -1 and 1 terminated by a *no-operation* (action=0). These policies focus on a longer escape trajectory of the valley floor, which achieves an optimal velocity at the transition between the unstable spiral and the limit cycle of the goal region such that the agent can coast to achieve the goal. In one respect, this policy is better in that it requires fewer changes in action. However, it would seem that these policies do not have low velocity in the goal region and are taking full advantage of the width of the goal region to achieve the goal state.

This analysis can be confirmed by examining the best policies in state-space, depicted in Figure 6.3(b). Figure 6.3(b) presents the state-space differently than the actual state-space. The boundaries of the plot have been changed such that the goal region has been shifted from the center of the state-space to the left. This places the boundary wrap (i.e., $x = 1.572$) more central in the figure so the unstable spiral of the problem's underlying attractor is more evident.

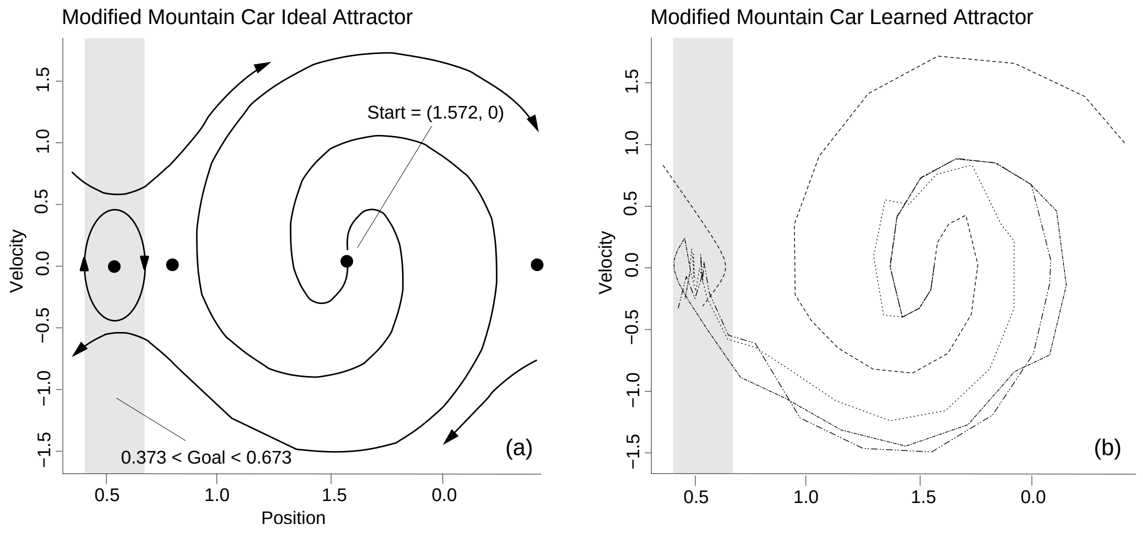

Figure 6.3: Modified Mountain Car attractors: a) ideal attractor structure; b) empirical attractor structure for the best policies.

The policies in Figure 6.3(b) are plotted as trajectories originating from $s_{init} = (1.572, 0)$. It is clear that two classes of trajectory exist for the best policies. One class, the first described above, minimizes the path length to reaching the goal region. Once inside the goal region, many small trajectory adjustments are made to ensure that the car stays within the confines of the reward region for 0.5 simulation seconds. The second class of trajectory follows the outer edge of the unstable spiral. The trajectory enters the goal region with high velocity and uses the system's intrinsic dynamics to slow the car. This trajectory is analogous to the second policy, described above, which reduces the car's velocity to remain within the goal region long enough to achieve the goal. Overall, however, the best policies fall well within the ideal attractor structure expected for the learned problem, given in Figure 6.3(a).

## 6.2 Acrobot

The ESN shows promise as a model for non-Markovian reinforcement learning domains in very low-dimensional space. Does the ESN architecture scale? To answer this question the ESN was applied to modeling the Acrobot Problem, presented graphically in Figure 6.5. Acrobot is dynamically equivalent to the double pendulum swing-up problem with con-

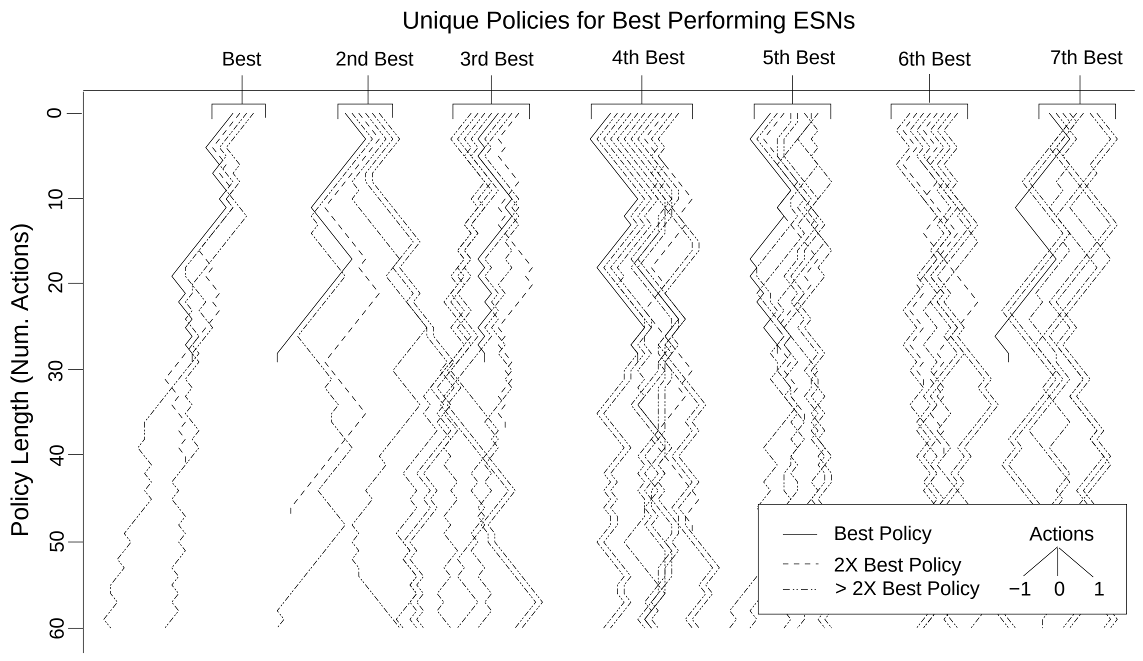

Figure 6.4: Unique policies for the 7 best performing ESNs for the Modified Mountain Car Problem. Note, ESNs are ranked from best (left) to 7th best (right).

straints placed on the action space. As with double pendulum, Acrobot consists of two masses attached via rigid links, having zero mass, at a joint which rotates 360 degrees. One link of the system is attached to a stationary joint which also rotates through 360 degrees. The double pendulum configuration always starts with the links parallel, pointing to the ground, with zero initial angular velocity—the dead hang position. Unlike some pendulum control problems, however, torque may only be applied to the system at the joint connecting the links and not at the base joint. This configuration simulates the actions of a gymnast on the high-beam apparatus, attempting to achieve a hand stand position from an initial dead hang by flexing muscles in the torso.

The double pendulum is a dynamically complex problem, known to behave chaotically [15]. Because of its simplicity of form, yet complexity of behavior, it has been well-studied as a control problem [15, 9]. Reinforcement learning of the swing-up problem has also been studied [72]. In this section the performance of the ESN as a model of the Acrobot domain is described and analyzed. First the system is described mathematically, followed by a description of the partially observable state-space, and particularly, the goal region

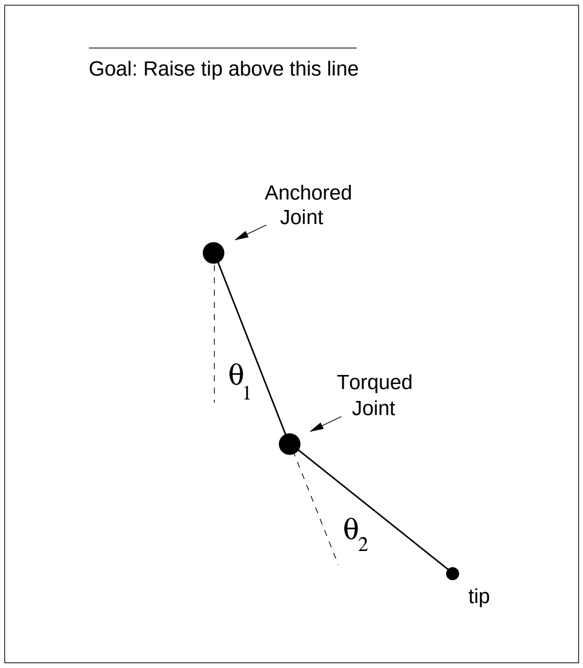

Figure 6.5: Diagram of the Acrobot Problem.

subset of this space. The section is concluded by a performance assessment of the ESN model of Acrobot.

### 6.2.1 Equations of Motion

The state-space of the Acrobot problem is $\mathbf{s} = \{\theta_1, \omega_1, \theta_2, \omega_2\}$ where $\theta_1$ and $\theta_2$ are the link angles and $\omega_1$ and $\omega_2$ are the respective angular velocities. The differential equations describing the motion of the Acrobot are:

$$\begin{aligned}
\frac{\partial \theta_1}{\partial t} &= \omega_1, && (6.1)\\
\frac{\partial \theta_2}{\partial t} &= \omega_2, && (6.2)\\
\frac{\partial \dot{\omega}_1}{\partial t} &= (l_2 m_1 g l_1 \sin(\theta_1) + l_2 m_2 g l_1 \sin(\theta_1) + l_1^2 m_2 \theta_2^2 l_2 \cos(\Delta)\sin(\Delta) + && (6.3)\\
&\quad m_2 l_1 l_2^2 \omega_1 \sin(\Delta))/(l_1^2 l_2(-m_2 - m_1 + m_2\cos^2(\Delta))), \text{ and}\\
\frac{\partial \dot{\omega}_2}{\partial t} &= (-l_1 l_2 m_1 m_2 g \cos(\Delta)\sin(\theta_1) + l_1 m_1 m_2 g l_2 \sin(\theta_2) - && (6.4)\\
&\quad l_1 l_2 m_2^2 g \cos(\Delta)\sin(\theta_1) + l_1 m_2^2 g l_2 \sin(\theta_2) - l_1 m_1 a -\\
&\quad l_1 m_2 a - m_1 m_2 l_1^2 \omega_1^2 l_2 \sin(\Delta) - m_2^2 l_1^2 \omega_1^2 l_2 \sin(\Delta) +\\
&\quad l_1 l_2^2 m_2^2 \omega_2^2 \cos(\Delta)\sin(\Delta))/(l_1 l_2^2 m_2(-m_2 - m_1 + m_2\cos(\Delta)^2))
\end{aligned}$$

where the following variables are defined,

$$\begin{aligned}\Delta &= \theta_1 - \theta_2, \\ l_1, l_2 &= \text{lengths of links, and} \\ m_1, m_2 &= \text{masses.}\end{aligned}$$

The parameter values used for Acrobot in this research are the same as those used in past experiments [72]. Link lengths and masses were each 1.0. Force of gravity was, $g = -9.8$. As described above, torque may only be applied to the central joint. The action space for the central joint was discretized such that $a = \{-1, 0, 1\}$. The problem was always initialized in the full dead-hang position with zero velocity, thus, $s_{init} = (\pi, 0, \pi, 0)$. The non-Markovian domain omitted velocity information. Therefore, $\tilde{s}_{init} = (\pi, \pi)$. The state-space was not wrapped, therefore, full inversion in observable state-space corresponds to any coupling of even multiples of $\pi$, $\tilde{s}_{inversion} = (p\pi, q\pi)$, where $p, q \in 2i, i \in \mathbb{Z}$. Likewise, the dead hang position in observable state-space corresponds to any coupling of odd multiples of $\pi$, $\tilde{s}_{deadhang} = (p\pi, q\pi)$, where $p, q \in 2i + 1, i \in \mathbb{Z}$. The goal of the problem is defined as rotating the tip of the outer link above 1.0.

### 6.2.2 ESN Learning Performance in the Acrobot Domain

While the complete state-space of Acrobot cannot be visualized, the partially observable state-space of link angles can be. The attractor underlying this system lies on a torus. The angle of the inner link carves a circle in state-space. Rotation of the outer link about this circle forms the surface of the torus. Any trajectory in observable state-space, if wrapped, lies on the surface of this torus. The physics of the problem gives intuition about how the learned policy should achieve the goal. The small amount of torque allowed to manipulate the central joint requires the agent to learn an oscillating policy of steadily increasing period. Therefore, at first, the policy should consist of high frequency action sequences alternating between -1 and 1. The frequency of these alternations should continuously decrease as the duration of each action sequence increases. This will build enough momentum for the system to rotate the tip of the outer link above a height of 1.0, which is equivalent to one full link-length above the central joint. The reward signal for achieving a goal state is 0, otherwise the reward signal is -1, making this a minimum path length decision problem.

Because the system is four-dimensional and the goal is a one-dimensional function of the observable state, an infinite number of goal states exist. Moreover, in observable state-space four different goal states exist that are equidistant from $\tilde{\mathbf{s}}_{init}$. From the perspective of a multistep decision problem, Acrobot is hard for several reasons: the state-space is four-dimensional—large for fine-grain lookup table representation; the dynamics of the system are chaotic; the small amount of torque applicable on any given step requires long policies to achieve the region; and the goal is a region, rather than a state, and is symmetric about the initial state-space.

### 6.2.3 Exploration and Reward Shaping in Acrobot

As described previously for MMCP, difficult learning problems sometimes require reward shaping to insure that the agent experiences goal states on a regular basis. The complexity of Acrobot makes achieving the goal state via random exploration of the state-action space very unlikely. For example, no goals were achieved using $\epsilon = 0.9$ for $500 \times 10^3$ training steps. This in turn makes modeling the Acrobot problem via function approximation particularly difficult, due to the lack of goal state experiences under learning schedules having high probabilities of exploration. To achieve the full learning problem, reward shaping was employed. The height of the goal state was initialized at 0.5 and then slowly raised to 1.0 according to the following schedule, $[5, 7, 9, 10]$, where changes in the goal height were invoked at training steps $[35 \times 10^3, 65 \times 10^3, 95 \times 10^3]$, respectively. These changes to the goal state were made in parallel to scheduled decreases in the probability of exploration, $[0.8, 0.5, 0.1, 0.05, 0.01, 0.005, 0.0]$, where changes were invoked at training steps $[35 \times 10^3, 65 \times 10^3, 90 \times 10^3, 110 \times 10^3, 130 \times 10^3, 150 \times 10^3]$.

### 6.2.4 ESN Modeling Performance in the Acrobot Domain

Preliminary experiments indicate that the reservoir parameters were sufficient to represent the Acrobot problem when the ESN size was increased to $N = 400$. The integration time-step of the leaky integrator logistic function was changed to $\Delta t = 0.2$ to match the integration time-step of the Acrobot simulation, which had an integration time-step of $\Delta t = 0.05$, but, unlike the MCP simulations, the Acrobot simulation followed the structure set down by Sutton [72]. These rules require that the simulation be iterated 4 times between each control decision with an effective $\Delta t = 0.2$. The finite horizon used to approximate the Q-values of this problem was 200. As in both the MCP and MMCP experiments, regression was used to train the linear readouts. Regression was performed every 1000 time-steps over a window of $20 \times 10^3$ echo states. Also, as in previous experiments, the regression was performed over the echo states and the squared echo states, added to ensure sufficient asymmetry of the reservoir's dynamics. Thus the total number of trainable parameters was $2 \times 404 = 808$. 150 learning trials were performed using this parametric configuration. The results of learning performance are summarized in Figure 6.6.

Figure 6.6(a) compares the mean ESN learning performance of all trials versus the 10% best performing ESNs. ESNs were ranked by the number of goals achieved under 100% exploitation (i.e., $\epsilon = 0.0$). Note, escapes were averaged over each 5000 training step segment of the learning trials. Unlike results presented for the MMCP, the distribution of ESN performance for the Acrobot problem is tight. The 10% best ESNs achieve maximum performance of 6.57 goals per thousand steps, equivalent to 152.2 steps per goal. In contrast, overall mean performance of all ESNs peaks at 5.3 goals per thousand training steps, equivalent to 188.7 steps per goal.

Figure 6.6(b) depicts the maximum best performance achieved by an ESN, which peaks at 9.2 goals achieved per thousand training steps equivalent to 109.0 steps per goal. The individual best goal achieving trajectory over all trials was 87 steps. The results for the overall mean ESN , best 10% mean ESN, and maximum ESN performances are best compared to results reported [72]. Sutton used an 18,648 parameter CMAC (i.e., multiresolution lookup table) to approximate the Acrobot using a complete representation of the state (i.e., both angles and angular velocities were accessible to the function approximations). This approximation converged to approximately 85-90 steps per goal.[3]

[3]These numbers are reconstructed from results in [72], which are reported graphically using a logarithmic

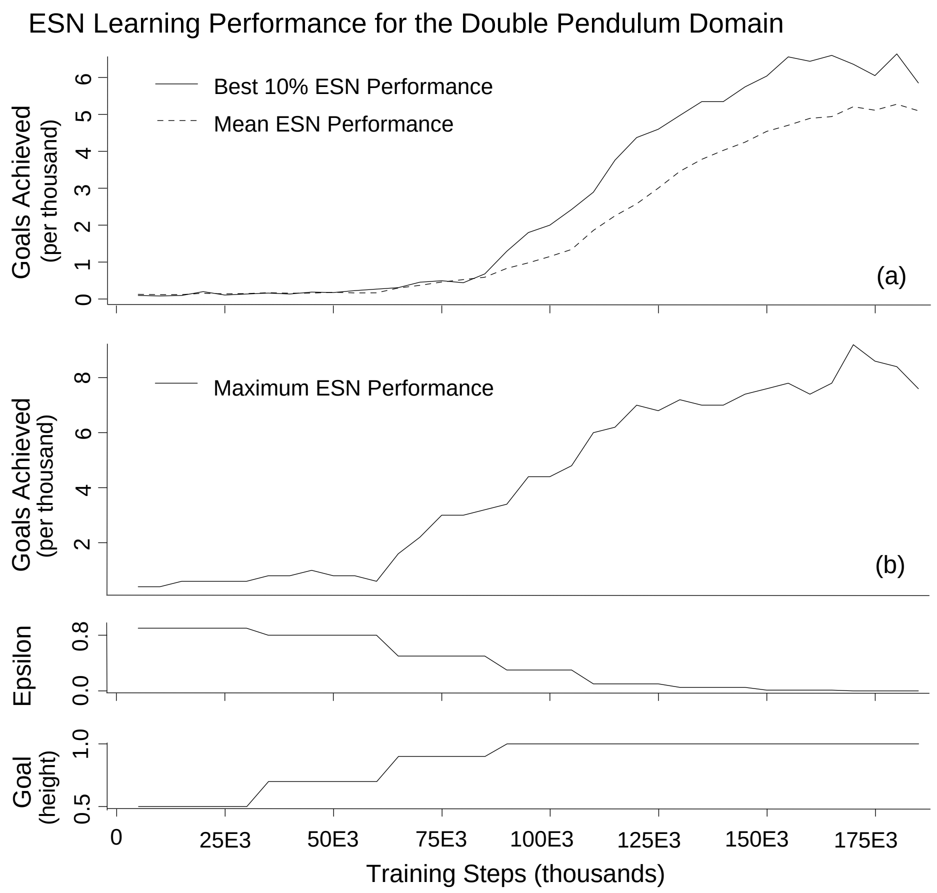

Figure 6.6: ESN modeling performance on Acrobot: a) overall mean and best 10% mean ESN performance; b) maximum ESN performance.

### 6.2.5 Analysis of ESN Learning Performance on Acrobot

This section attempts to build an understanding of the structure of the policy learned by the ESN and to provide some insight into why the ESN is capable of representing this system, despite its complexity. To do this the characteristics of the best performing ESN over all 150 trials in both policy space, Figure 6.7, and state-space, Figure 6.8 are analyzed.

In Figure 6.7 the policy (i.e., trajectory) length, policy, learning schedule, and goal schedule are plotted in parrallel through time. In Figure 6.7(a) a dashed line was included, which marks the approximate policy length found via CMAC with complete state information [72]. This line is assumed to be the near-optimal solution to the system. The purpose

dependent axis.

of these plots is to demonstrate several features of ESN learning when the dynamics of the reservoir are well-suited to represent the domain. For orientation a few features of Figure 6.7 must be pointed out. First, the independent axis is plotted in trials, rather than learning steps. This accounts for the steep increase in both the learning and goal schedules seen in the lower plots of Figure 6.7. Trials only restart if the agent achieves the goal or the 5,000 learning step limit is reached. Therefore, it is reasonable to expect that the vast majority of goals are achieved only after the learning schedule favors exploitation of the learned Q-values. Second, the policies in Figure 6.7(b) are plotted similarly to the policies in Figure 6.4 as tree traversals. Diagonal lines (left and right), correspond to actions -1 and 1, respectively. Vertical lines indicate a no-op (action=0). Time runs from the root of the traversal to the leaf. For plotting purposes, policies were truncated to lengths of 150 actions. Third, it should be noted that the upper performance plot is presented with a logarithmic dependent axis.

The primary result presented in Figure 6.7 is evidence that the ESN is capable of near-optimal performance in the Acrobot domain given only partial state information. Trials 280, and 560—576, exhibit policy lengths of 87 and 94 steps, respectively. This performance is in the range of the best known learned performance on Acrobot [72, 9], and is achieved within 600 learning trials, very similar to the 500 learning trials required for the CMACS approximation to converge.

Second, policy plots provide useful insights into the learning behavior of the ESN. Of these, the most evident feature is empirical confirmation of anticipated behavior of learning via regression. Randomness injected into the policies during early trials persists in the ESN model long after the learning schedule has transitioned to an exploitation regime. Visual inspection of the policy plots for the first 20 actions shows an emergent structure as the learning schedule trends toward $\epsilon = 0.0$. At this point in the learning schedule long policy structure develops, up to approximately 40 actions, but disorder in the policies persists until approximately the 475th trial, after which policies exhibit primarily parallel structure out to 100 actions. The entrainment of random actions into the echo state is a serious weakness of the ESN and does not, at present, have a solution.

An additional benefit of visualizing policies as tree traversals is the ease with which policies may be compared to the best policies in the exploitation regime. The structure of the highly efficient policies of trials 560—576 may be easily compared to that of their neighbors, and discrepancies, particularly early in the policies, may be identified as cracks in the structure of the plot. The decision point, and action selection, where suboptimal policies diverge from the best policies is easily identified, as are temporal structural trends in what the ESN has learned, both in the space of policies and temporal evolution of trials. Of these, the result most striking is the sub-optimality of policies in the exploitation regime. Sub-optimality begins early in the action selection process. Structural differences occur within the first 10-15 actions, after which the policy structure mimics that of the best policies. The sensitivity of the action space is apparent in that these small divergences from the best policies cause the policy length to increase greatly. This implies that small errors early in the policy are important (i.e., the goal state sought by the best policies was not achieved by this sub-optimal policy). This type of behavior is highly indicative of bifurcations in the learned attractor as was discussed in Chapters 4 and 5.

The structure of the policies at any $\epsilon = 0$ point in the learning trial depicts exactly what the ESN has learned to that point. Polices, however, do not provide intuition into how the agent is interacting with the system in the state-space. This intuition must come from an understanding of the how the policy influences the attractor underlying the learning problem. Because the Acrobot is a 4-dimensional attractor it cannot be viewed directly. Instead, we must rely on partial representations of the attractor, such as Figure 6.8.

Figure 6.8 depicts three plots of the attractor underlying the Acrobot learning problem in partially observable state-space (angles only) in three regimes of the learning schedule. Figure 6.8(a) depicts all policies followed by the ESN when the goal height was set to 1.0 (the full learning problem). Note, gray circles in all plots indicate the goal regions. Figure 6.8(b) depicts a subset of the policies in Figure 6.8(a) where $\epsilon = 0$ (i.e., complete exploitation). Finally, Figure 6.8(c) depicts the two best policies (87 and 94 steps, respectively) in state-space.

The contrast between Figures 6.8(a) and (b), shows us that the ESN is capturing es-

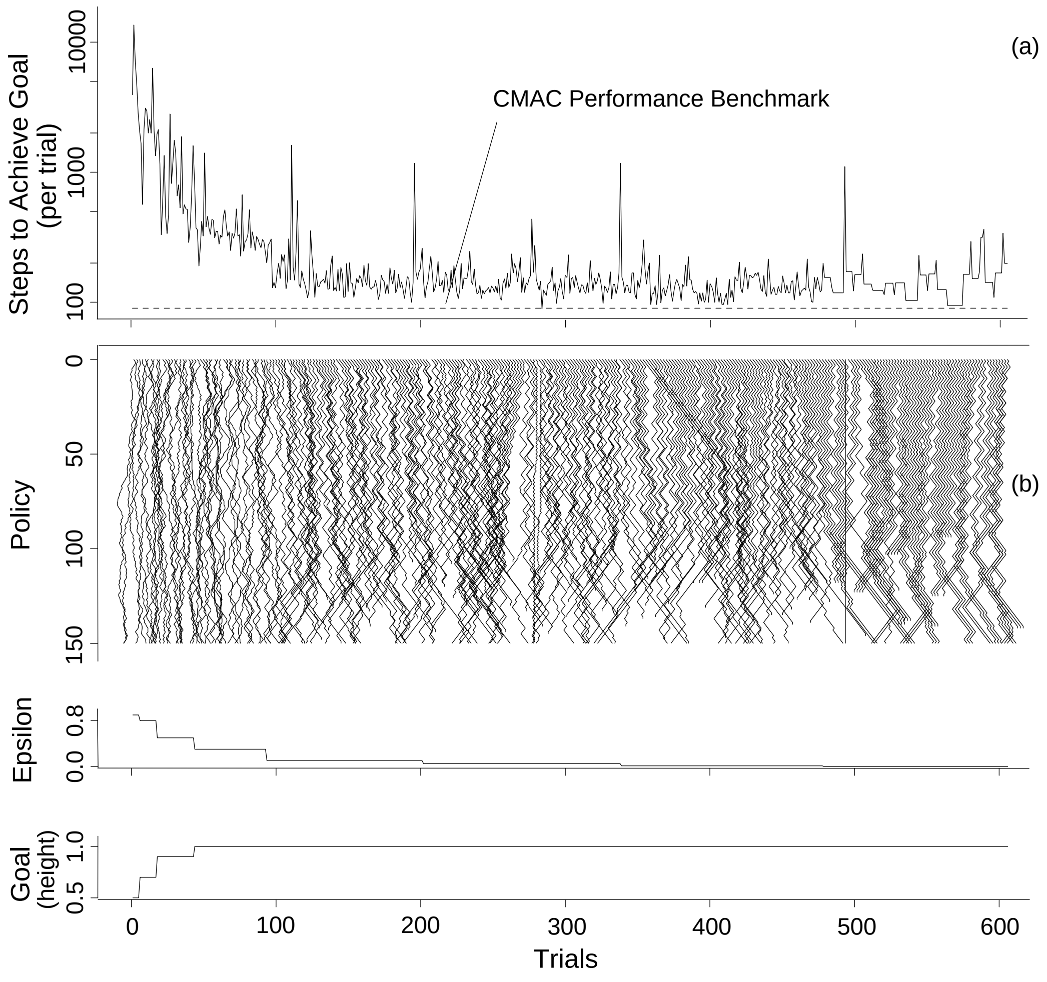

Figure 6.7: Best ESN performance and policy versus learning trials on Acrobot: a) performance for each trial in the learning sequence and b) policy used for each trial.

sential state-space information. As exploitation of the ESN increases, the policies learned by the ESN avoid large portions of the state-space, constraining the agent to a region of $\theta_1 = [0, 2\pi]$ and $\theta_2 = [-8\pi, 10\pi]$. As is evident from both Figures 6.8(a) and (b), the state space of the outer link is explored considerably more than that of the inner link, a result of the system's dynamic constraints. The Acrobot contains no friction to oppose the rotation of the joints. Over time, adding torque to the system will achieve very high velocities for the outer link, and, subsequently, the inner link. Once the links attain high rotational velocities, the odds of achieving a goal state increase. This is important for achieving the goal state when the probability of random exploration is high, but is detrimental to learning efficient policies. The agent does, however, learn to ignore high angle regimes and focus on the state-space immediately surrounding the start state, $\mathbf{s}_{init} = (\pi, \pi)$, a benefit of using a minimum path reward structure.

Figure 6.8(c) provides insight into the structure of the attractor underlying the Acrobot domain. The two shortest policies learned by this ESN after 500 trials constrain the Acrobot to an unstable spiral structure in the partial state space $\tilde{\mathbf{s}} = (\theta_1, \theta_2)$. The best policy finds a goal region without leaving the state-space range $[0, \pi]$. Cross-referencing this state-space information with the policy depicted in Figure 6.7 at trials 560—576, provides a visualization of what is learned. This policy is an alternating sequence of -1 and 1 subsequences, where the frequency of alternation increases in time. This policy is exactly what would be expected for this problem. Alternating torque on the central joint would cause the inner link to rock back and forth. As maximum angle of the inner link's swing increases, torque must be applied to the central joint for longer periods of time. Eventually, a sufficient deflection of the inner link allows the tip of the outer link to rotate above the goal height.

Because the structure of the incomplete state-space is an unstable spiral for the best policy, it is possible to determine the structure of the complete state-space and construct the attractor underlying this learning problem. Based on the unstable spiral structure of the angular space, the angular velocities as functions of time will be two out of phase oscillations that grow in magnitude. The angular velocity in the phase plane will also be an unstable spiral structure. Therefore, the attractor underlying the Acrobot reinforcement learning domain is a 4-dimensional unstable spiral, which confirms previously reported results [9]. This motivates how the ESN architecture well-models this problem and why the distribution of performances for the ESN architectures are tightly coupled. While high-dimensional, the 4-dimensional unstable spiral exhibits only one dynamic component, which exists at only one scale. As is evident in lower dimensional experiments, the ESN architecture is well-suited to representing the unstable spiral attractor at low-dimension. Higher dimensions, while making the attractor harder to distinguish based on experience, once found, is well-modeled by the ESN architecture.

The results in this chapter provide some insight into ESN learning of the full inverted variant of Acrobot (i.e., where the goal is defined as $\parallel \tilde{s} - \tilde{s}_{inversion} \parallel < \epsilon$ for some small value,

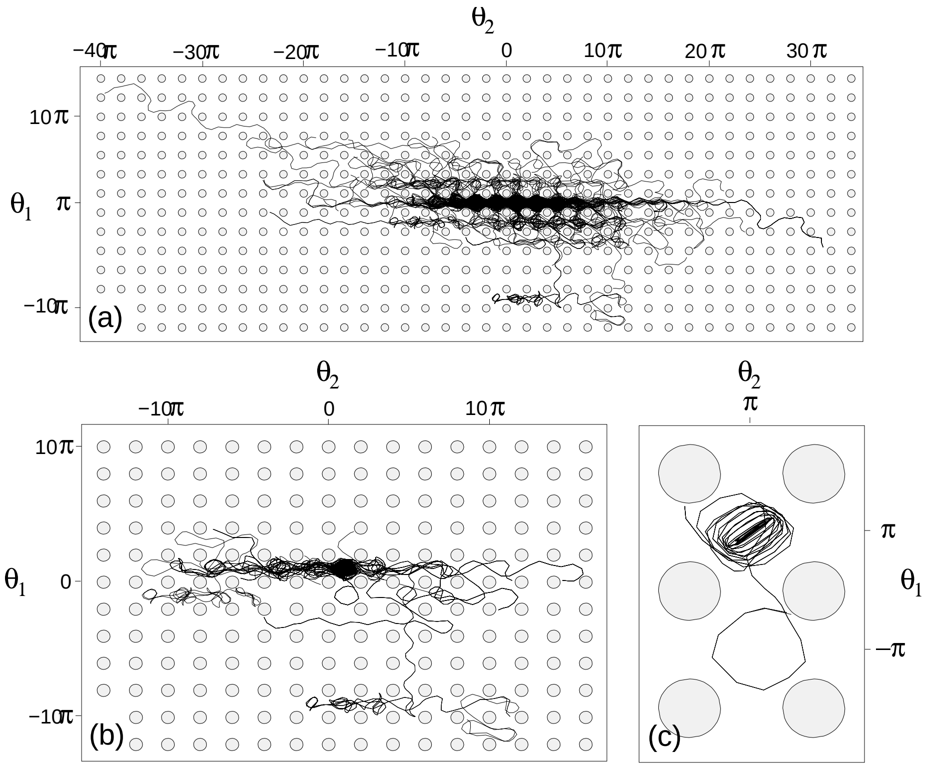

Figure 6.8: Observed trajectories of the learned Acrobot domain for the best ESN: a) All trajectories in which goal height was 1.0; b) All trajectory for which $\epsilon = 0.0$; and c) two best trajectories having lengths of 87 and 94 actions, respectively.

$\epsilon$). It would be expected that this problem is extremely difficult for the ESN to model using the basic training method defined in this research. This difficulty is anticipated because the inverted variant (swing-up and balance) would require the ESN to represent two attractor components, an unstable spiral and a limit cycle, at two different scales of control granularity in 4-dimensional space. While capable of doing this, it is very difficult to envision a training algorithm that would succeed in building sufficient experience for the ESN to develop this attractor. Thoughts on how this might be achieved are relegated to Chapter 7 and 8 where alternate training techniques for the ESN architecture are reviewed.

# Chapter 7

# Mixtures of Readouts

ESN predictions are linear mappings of the *reservoir*. If the reservoir is not initialized with dynamics appropriate for the target system, the well-known "reservoir richness" problem, then linear mappings of this basis will fail to accurately model the target system. Without adaptation the reservoir is altered only through random reinitialization. Complex domains, however, may require hundreds or thousands of reinitializations, and in some cases, no suitable reservoir may be found [61]. The reservoir richness problem also occurs in the modeling of reinforcement learning domains, evident by the poor modeling performance of the ESN on the low-dimensional and dynamically simple Modified Mountain Car domain, described in Chapter 6.

To overcome this limitation, a body of literature devoted to machine learning and signal processing techniques for reservoir design and adaptation now exists: reservoir design with *a priori* knowledge of target dynamics [30], linearization and pole analysis [56], spectral radius tuning via input bias adaptation [56], reservoir topology optimization via next ascent local search [11], reservoir adaptation via intrinsic plasticity [70], and adaptation of global reservoir parameters via gradient descent [36].

All of these techniques, however, directly or indirectly manipulate the dynamics of the reservoir. For instance, the intrinsic plasticity rule and gradient descent adapt the reservoir topology and connection weights, either online or batchwise. Alternatively, search requires batchwise alteration of reservoir topology and recomputation of reservoir dynamics to determine fitness. These approaches, unfortunately, circumvent the original echo state approach's greatest advantage over adapted recurrent networks—the reservoir's readouts may be trained via linear regression [30].

One way to bridge the disconnect between efficient training of linear readouts and the need for reservoir adaptation is through decomposition of the ESN training problem [13]. A way to decompose the problem is to temporally embed the state-space of the target system and identify clusters in this embedded space. Linear readouts can then be trained via regression to fit separate dynamic subsystems to the samples in each cluster. This insight links ESN training with the mixture of experts (MoE) framework—a hierarchical modeling approach in which the influence of local *expert* approximations are mixed based on higher-level features of the state space. MoE includes both unsupervised and supervised learning techniques [29, 37]. The replacement of the ESN readout layer by a MoE is termed mixture of readouts (MoR). This distinction is for clarity in the literature rather than a claim of novelty.

This chapter contributes to the reservoir adaptation literature in several ways. The training of reservoir readouts into the MoR framework is formally recast. This framework permits adaptation of the projection of the target system onto the reservoir while preserving the ability to train the readouts via linear regression. This work also demonstrates how unsupervised modeling of the reservoir's state-space can exploit previously unused information to improve prediction performance with minimal increase in computational overhead. This work also connects reservoir computing with earlier application of MoE to time-series prediction [5], the primary contribution of this work being the replacement of embedded temporal feature vectors with the reservoir's state-space. While the technique is developed and demonstrated for stationary attractors, it may be possible to apply this technique to nonstationary attractors, such as those found in reinforcement learning domains.

The remainder of this chapter is organized as follows. First, the MoR framework, training algorithm, and motivation are discussed. Next, performance results are presented for the MoR applied to the Lorenz and Mackey-Glass chaotic dynamic systems. The chapter is closed with comments on the performance potential of the MoR approach in modeling nonstationary reinforcement learning domains.

## 7.1 Mixture of Readouts

Mixture of readouts is the reservoir computing analog to unsupervised mixture of experts (MoE) [29, 37]. The primary contribution of MoR is its applicability to efficient learning of predictive models. In this section the MoR framework is developed assuming a two level hierarchy. The lower level contains the linear readouts acting as local experts. The higher level contains the mixture model which determines the strength (or mixture) of expertise needed to predict the next system state. Generalization to multilayer hierarchies, multiple reservoirs, other reservoir implementations, and supervised mixture models are easily accommodated.

### 7.1.1 The MoR Framework

Consider a discrete dynamic system having state vector $\mathbf{s}(t)$ that is to be modeled via MoR. The MoR consists of a *reservoir*, *readout*, and *mixture model*, defined below.

Reservoir: As described in Chapter 3, the ESN reservoir is a stochastically-generated, sparsely-connected recurrent network, which stores a rich, dynamic feature-set of system history. The reservoir has an internal hidden state, $\mathbf{x}(t)$, of dimension $N$ and contains three separate connection topologies: $\mathbf{W}_{in}$ of size $N \times M + 1$ is a mapping from the input, $\mathbf{u}(t)$, onto the hidden state where $\mathbf{u}(t) = [\ \mathbf{s}(t),\ b\ ]$, $\mathbf{s}(t)$ is the system state at time $t$, having size $M$, and $b$ is the scalar input bias; $\mathbf{W}_{hid}$ of size $N \times N$ is a recurrent mapping within the hidden state; and $\mathbf{W}_{back}$ of size $N \times M$ is a recurrent mapping from the predicted future system state, $\tilde{\mathbf{s}}(t)$, onto the hidden state.

The reservoir update is defined by the recurrence in Equation 3.9, where activation function $f(\cdot)$ is the logistic function or a leaky integrator [30, 36]. Reservoir topology is configured by parameters $\alpha$ and $\rho$ in addition to dimension $N$. The scaling factor, $\alpha$, when applied as a scalar multiple of $\mathbf{W}_{hid}$ guarantees global stability [30] if $\alpha < 1/ \mid \lambda_{max} \mid$ where $\mid \lambda_{max} \mid$ is the maximum eigenvalue magnitude of $\mathbf{W}_{hid}$. This condition guarantees that point-wise separability of the dynamics approaches zero independent of the input sequence. The density parameter, $\rho$, determines the percentage of nonzero entries in $\mathbf{W}_{hid}$.

Readout: The readout consists of a set, $\mathbf{W}$, of $K$ readout matrices, $\mathbf{W}_k$, $k \in K$, each of

size $M \times 2M + N + 1$. Each readout matrix defines a linear mapping from the embellished reservoir state, $\mathbf{z}(t) = [\ \mathbf{u}(t),\ \mathbf{x}(t),\ \tilde{\mathbf{s}}(t-1)\ ]$, onto the predicted future state, $\tilde{\mathbf{s}}(t)$.
Mixture Model: The mixture model is a function, $\mathcal{M}$, mapping a feature vector, $\mathbf{v}(t)$, onto a vector of mixture coefficients, $\mathbf{m}(t) = \mathcal{M}(\mathbf{v}(t))$. Vector $\mathbf{m}(t)$ is comprised of $K$ elements, $\mathbf{m}_k(t)$, $k \in K$. Each $\mathbf{m}_k(t)$ is a scalar weight corresponding to the $k$th readout, $\mathbf{W}_k$.

MoR prediction of future state for each time $t$, therefore, is a weighted sum over the products of the readout matrices and embellished reservoir state vectors,

$$\tilde{\mathbf{s}}(t) = \sum_{k \in K} \mathbf{m}_k(t) \mathbf{W}_k \mathbf{z}\,(t)\,. \tag{7.1}$$

### 7.1.2 Training the Readouts via Unsupervised Mixture Model

The ESN is trained analytically [30] by regression over the system of linear equations,

$$\mathbf{Z}\mathbf{W}_{out} = \mathbf{S}. \tag{7.2}$$

$\mathbf{W}_{out}$ is the ESN's trainable linear readout. Each row of the model matrix, $\mathbf{Z}$, is the embellished reservoir state vector, $\mathbf{z}(t)$, $t = 1,...,T$ where $T$ is the number of samples in the training dataset. Each row of the constraint matrix, $\mathbf{S}$, is the system state vector $\mathbf{s}(t)$, $t = 1,...,T$. This simple, efficient training method is what makes the ESN, and reservoir computing in general, so attractive for predictive modeling.

When generalizing the ESN to MoR, however, depending on the type of mixture model used, the training method can change. When the mixture model is trained by supervised learning, the linear readouts cannot be trained by regression because the readout matrices, $\mathbf{W}_k$, and the mixture coefficients, $\mathbf{m}_k$, are no longer independent. In this case it is necessary to train the readouts via gradient descent or a statistical process. An unsupervised mixture model, however, preserves the training problem presented in Equation 7.2. In this case, the mixture weights and the readout weights are independent. Therefore, Equation 7.2 can be modified from a single regression to that of $K$ regressions over the system of linear equations,

$$\tilde{\mathbf{Z}}_k \mathbf{W}_k = \tilde{\mathbf{S}}_k. \tag{7.3}$$

$\tilde{\mathbf{Z}}_k$ is a matrix of size $T \times 2M + N + 1$ having rows, $\mathbf{m}_k(t)\mathbf{z}(t)$, $t = 1,...,T$. $\tilde{\mathbf{S}}_k$ is a matrix of size $T \times M$ having rows, $\mathbf{m}_k(t)\mathbf{s}(t)$, $t = 1,...,T$.

### 7.1.3 Motivation

A mathematical framework was developed above that adds complexity to both the ESN readout and the training algorithm. To justify adding complexity insight is needed into how the echo state reservoir interacts with system history, how this behavior limits prediction performance, why this limitation occurs systematically during ESN construction, and how MoR decreases these deficiencies. This section builds this justification.

For this discussion an actual MoR example will be used, which is summarized in Figure 7.1. The Mackey-Glass time-series (detailed in the experimental methods section) shown in Figure 7.1(a) is modeled via ESN using Equation 3.11. Ten ESNs are trained in this way and the squared training errors are depicted as time-series in Figure 7.1(c). These time-series are summed in Figure 7.1(b) to illustrate correlations between the individual sequences.

There are three important features of Figures 7.1(a,b, & c). First, strong correlations exist between the error sequences. Second, these temporal correlations are roughly consistent with periodic (and half-periodic) behavior observable in the time-series. Finally, the variance of temporal errors across individual sequences is relatively small. Thus, it can be concluded that randomly generated ESNs, once trained, consistently fail to approximate specific locations in the time-series' underlying attractor. This aspect of ESN training has been observed across numerous dynamic systems.

The next step of our reasoning requires the transformation of temporal analysis to spatial analysis. The original time-series can be embedded to build a higher dimensional state-space that, according to Takens Theorem [74], preserves the original system dynamics. The example has an embedding dimension of 13, which means that each embedded state is a 13-dimensional projection of consecutive points in the time-series. To visualize this state-space a Sammon mapping [66] is constructed. The Sammon mapping is a dimensionality reduction technique that preserves relative distances between points as dimensions are removed. Figure 7.1(d) depicts the embedded state space of one error sequence from

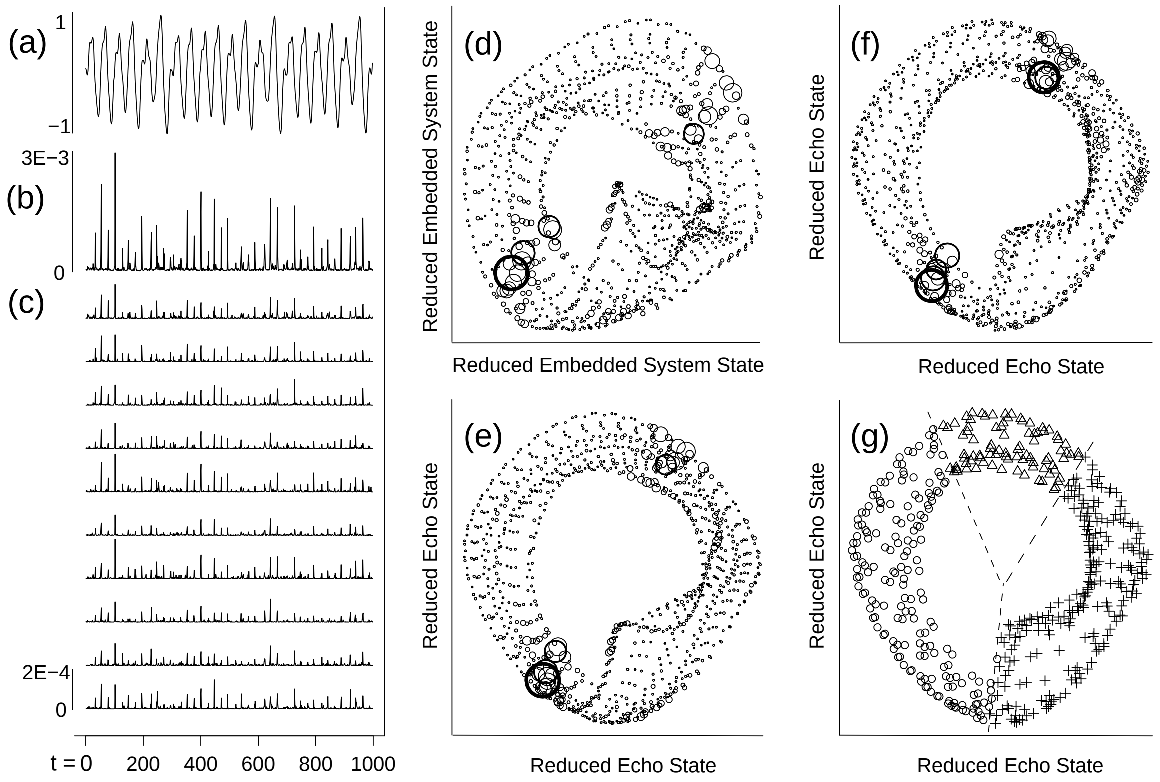

Figure 7.1: Conceptual diagram of mixture of readouts (MoR): (a) time-series, (b,c) temporal sequences of ESN training error, (d) Sammon map of the system's embedded attractor, scaled by training error, (e) Sammon map of system's echo state attractor, scaled by training error, (f) Sammon map of the system's echo state attractor, scaled by open-loop generalization error, and (g) example $K$-mean trained mixture model. Voronoi regions are separated by a dashed line.

Figure 7.1(c) projected down to two dimensions. The size and line-thickness of the circles in Figure 7.1(d) are scaled by the magnitude of error in the original sequence. It can be seen that, after training, two specific regions of the attractor contain the majority of error. This is another aspect of ESN training, which is consistent across dynamic systems—the reservoir often is not rich enough to represent the underlying attractor.

This claim can be supported by directly observing which echo states contain the training error. To do this ESN theory is consulted. The embellished echo state, $\mathbf{z}(t)$, represents a linear projection of the system's attractor. The echoes represent an approximate embedding of the time-series and preserve the dynamics of the attractor's state-space. The dimensionality of the echo states can be reduced via Sammon map and label the points by their training error, as shown in Figure 7.1(e). This figure graphically presents two observations seen over numerous dynamic systems and ESN configurations. First, the echo states do, in fact, approximate an embedding of the system's attractor. Second, ESN training localizes error into dynamically compact regions of the attractor.

Why do these errors occur and why do they cluster? Regression error results from representing a complex system with a simpler model—this is also true of regression over complex attractors. The probability that a random reservoir can linearly model complex dynamics is low. More likely, certain dynamics of the attractor can be modeled with little error. Other dynamics will contribute varying degrees of error to the linear fit. This explains the existence of error but not its locality.

Complex attractors are generally composed of many pieces. Each of these pieces exhibit simple dynamics but combine to form complex global dynamics. Attractor complexity, therefore, is relative to the point of observation. Seemingly complex dynamics, when viewed from any single, local observation point, may seem trivial. Taking this argument one step further, an attractor would be simple if its complex dynamics were viewed piecewise as a sequence of locally simple dynamic systems. Dynamic locality can be observed by comparing the upper and lower portions of the attractor depicted in Figure 7.1(d).

As described above, each dynamic component of the attractor will contribute some amount of error of the linear fit. Dynamics of the attractor that are poorly modeled by the reservoir will contribute the greatest amount of error. Because of dynamic locality, the points of the attractor exhibiting these dynamics will be relatively close together. Therefore, the error associated with these points will also lie close together on the attractor. In addition, sources of training error generally correspond to sources of generalization error (i.e., testing error, either open or closed loop) having similar or greater severity. An example of this artifact is shown in Figure 7.1(f).

These error clusters can be removed by exploiting dynamic locality—the MoR approach. Using a mixture model to partition the attractor enables the regression problem to be broken into $K$ dynamically local regressions. This reduces the number of constraints that regression must satisfy to project the system onto the reservoir. In turn this increases the probability that a random reservoir will exhibit dynamics suitable to the regression problem. In short, one complex regression problem is replaced with $K$ local regressions. As an illustration, a $K$-means (details below) mixture model partitioning is shown in Figure 7.1(g). Each local dynamic component is plotted using a unique symbol.

### 7.1.4 Training the Mixture Model

The focus of this chapter is training of a Gaussian mixture model of the reservoir's dynamics via the expectation maximization (EM) method. Also studied is the $K$-means method, a special case of EM, which has particularly efficient training properties. Our interest in unsupervised models is the identification of gross characteristics of target system dynamics. One difference between EM and $K$-means, for our purposes, is the interpretation of boundaries between clusters. Because $K$-means enforces strict set assignment, this restriction can be interpreted as a hard boundary. In contrast, the continuous probabilistic membership assignment of the Gaussian mixture model acts as a soft boundary where a data point may belong to more than one set. The trade-off between hard and soft boundaries is a matter of performance versus overfitting; $K$-means clustering is computationally inexpensive compared to EM but is more likely to over fit the training data.

Expectation maximization (EM) [16] is an unsupervised modeling technique which, assuming a dataset is probabilistically distributed, finds maximum likelihood parameters for a set of $K$ multivariate models. Each of the $K$ models has an associated distribution and weight. The technique utilizes two steps, an, expectation, $E$-step, which computes an expectation of the likelihood that the model fits the data, and a maximization, $S$-step, which modifies model parameters to maximize the expectation of the $E$-step. The $E$ and $M$ steps are repeatedly iteratively until model likelihood converges. Given a feature of the state-space (e.g. a time-series embedding or an echo state), the EM algorithm, computes a mixture vector.

$K$-means is a special case of the EM technique. $K$-means assumes that the dataset is drawn from a set of $K$ multivariate Gaussians, which have covariance matrices equal to the identity matrix. The weights of all Gaussians are also assumed to be equal. These restrictions enforce strict set assignment. Each data point must belong to one and only one of the $K$ Gaussian models. The mixture vector of $K$-means, therefore, is binary valued.

In fact, $K$-means training is $K$ times cheaper than EM. This is possible because of the mixture coefficients of hard partitions are binary. The linear regression of Equation 7.3 can be decomposed into $K$ linear regressions. Because the matrices $\tilde{\mathbf{Z}}_k$ and $\tilde{\mathbf{S}}_k$ are binary valued, the individual problems can be condensed to those portions corresponding to mixture values of 1.0. This effectively makes expense of building the MoR for $K$-means less than or equal to that of Equation 3.11 plus the cost of building the mixture model.

## 7.2 Experiments

The MoR approach is explored empirically by its performance on two well-known chaotic systems: Lorenz and Mackey-Glass, $\tau = 17$. Of particular interest are the arguments for and against higher levels of mixture model complexity, particularly when the reservoir's echo state is the feature vector used to learn the mixture model.

MoR performance is measured by closed-loop iterations. However, our definition of a closed-loop iteration is more stringent than previous experiments [61, 13, 30] and is computed as follows. The trained reservoir is brought into equilibrium by teacher forced iteration over the training sequence [30]. At this point in past experiments [61, 13, 30], A single, closed-loop run over the test sequence was performed, terminating when the squared prediction error exceeded a predefined threshold. The performance was defined as the number of iterations completed before the threshold is exceeded. This method, however, generated high variance. Using closed-loop disturbances decreases the variance in our performance measurements. Closed-loop disturbances are defined below.

The state of the reservoir is recorded at the end of equilibration. At the first closed-loop prediction step a uniformly distributed disturbance on the range [-$\epsilon$, $\epsilon$] is added to the input. The closed-loop is then run freely to termination. The system is reset to the end of equilibration, randomly disturbed, and rerun. The mean and variance from a single reservoir, measured over multiple, disturbed closed-loop runs comprise a data point. These measurements were then repeated multiple times over the ESN parameter configuration to determine performance for that class of ESN.

### 7.2.1 Case Study: Lorenz

To test MoR on a multidimensional state-space its prediction performance on the 3-dimensional Lorenz attractor is studied. The attractor was generated by fourth order Runge-Kutta numerical integration of the Equations:

$$\begin{aligned} \dot{x} &= \sigma(x-y) \\ \dot{y} &= x(\rho-z)-y \\ \dot{z} &= xy-\beta z \end{aligned}$$

where $\rho = 28$, $\sigma = 10$, $\beta = 8/3$, $\Delta t = 0.01$. MoR performance was tested for a range of reservoir sizes, $N$, and number of model components, $K$, using reservoir dynamics similar to previous Lorenz studies [35]. Consistent reservoir dynamics were enforced across reservoirs of various size $N$ by scaling $\rho$. This scaling ensured that the average number of connections to each element of $\mathbf{x}(t)$ remained constant as $N$ varied. Performance results ($threshold = 0.01$ and $\epsilon = 0.002$) averaged over 30 disturbances and 30 reinitializations, using a $K$-means mixture model, are presented in Figure 7.2(a).

Prediction performance for all reservoirs improved (by $\approx 63\%$ for the largest reservoir, $N = 500$, and up to $\approx 525\%$ for $N = 75$) as $K$ varied. All MoRs improved for $K = 2$ model components, which is not surprising. Examination of a 2-dimensional Sammon map of an example ($N = 100$) echo state trajectory, Figure 7.2(b), shows a decomposition into two ellipsoidal distributions, which are well-modeled by Gaussians. Points in the mapping are plotted with symbols representing model membership (a dashed line divides the Voronoi regions). MoR clusters the distributions intuitively. It can also be observed that one of the partitions absorbed the central state-space, which is critical to $K$-means mixture model success because it prevents overfitting at the attractor's transition point.

### 7.2.2 Case Study: Mackey-Glass, $\tau = 17$

The Lorenz attractor is a system that is well-represented by a Gaussian mixture model. This is not true of all attractors. This study works with the Mackey-Glass (MG), $\tau = 17$,

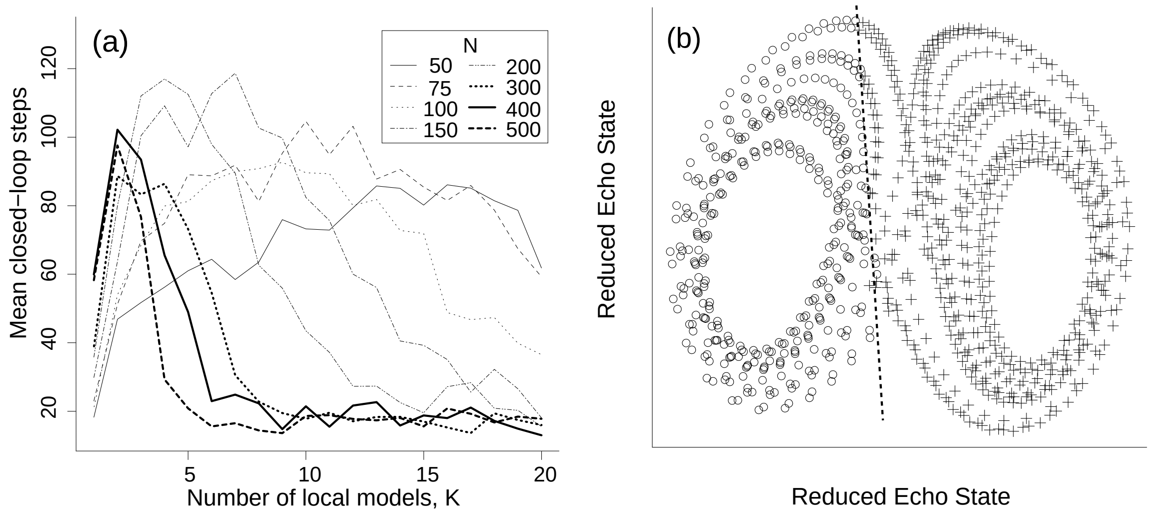

Figure 7.2: MoR modeling of the Lorenz attractor: (a) MoR generalization error of $K$-means trained echo-state mixture model as ESN size, $N$, and model components, $K$, vary; (b) Sammon map of the echo state with partition boundaries, $K = 2$. Voronoi regions are separated by a dashed line.

attractor—a system that is not cleanly represented by a Gaussian mixture model. Our MG attractor was generated via 4th-order Runge-Kutta numerical integration of the system of equations:

$$\dot{x} = \alpha \frac{x(t-\tau)}{(1 + x(t-\tau)^{\beta})} - \gamma x(t)$$

ESN dynamics and parameters were identical to those used in previous ESN experiments on MG [30] corresponding to best known performance for a single reservoir: $\alpha = 0.2$, $\beta = 10$, $\gamma = 0.1$, $\tau = 30$, $\Delta t = 0.1$ and $x_0 = [0, 0.2]$.

The attractor was studied with two experiments. First, MoR was performed using a $K$-means mixture model that takes an embedding, $E = 13$, of consecutive system states as the feature vector, which corresponds to the motivational example in Figure 7.1(d). MoR performance from this configuration for a range of reservoir sizes and number of model components is summarized in Figure 7.3(a). These results have several interesting features: performance increases dramatically for the smallest reservoir, no performance improvement occurs for the largest reservoirs, and for all reservoir sizes a precipitous drop using $K = 2$ mixture model components is observed.

Figures 7.1(d & e) aid in understanding this drop in performance. The difference be-

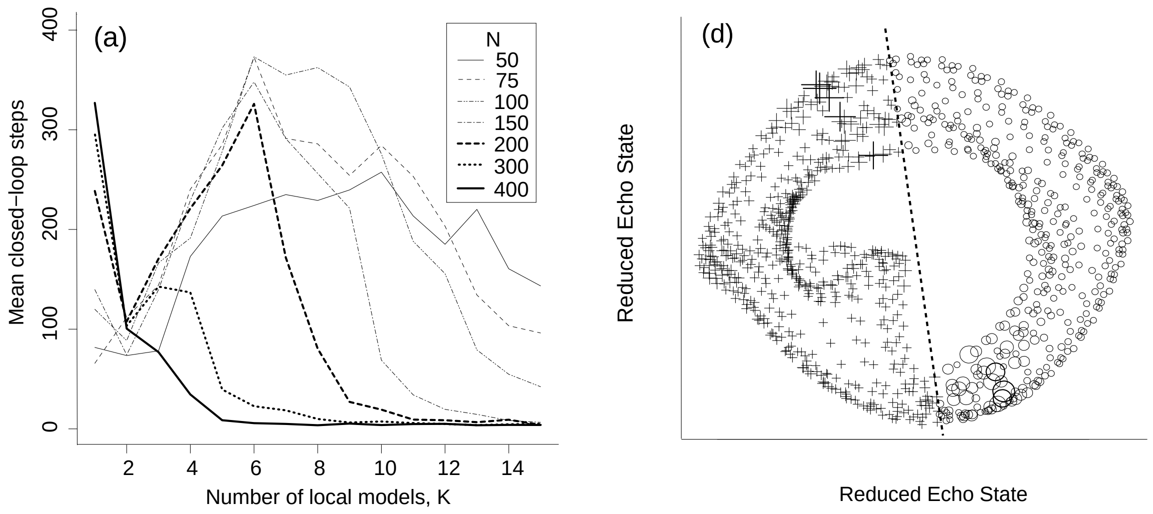

Figure 7.3: MoR modeling of the Mackey-Glass attractor: (a) MoR generalization error of $K$-means trained embedded state-space mixture model as ESN size, $N$, and model components, $K$, vary; (b) Sammon map of the embedded system state with partition boundaries, $K = 2$. Voronoi regions are separated by a dashed line.

tween the embedded state and echo state attractor is one of detail. Dynamics of the system are preserved in both attractors, but the details of the state-space dynamics are blurred by the echo state attractor. Lack of dynamic detail in the echo state representation allows unsupervised partitions to cut across locally homogeneous dynamics of the underlying attractor—an incorrect partition. Because the ESN reservoir is randomly generated, however, the partition localizations vary over trials. Poor localization increases generalization error.

The severity of performance decrease at $K = 2$ is an artifact of the MG attractor and is not a general feature of MoR. This explanation is illustrated in Figure 7.3(b). The embedded states have been plotted with symbols according to their partition membership. The sizes of the symbols are proportional to the generalization error of the point when $K = 1$. When partitioning MG into two dynamic regimes, the partition boundary, unfortunately, aligns with errors localized during regression. This is the worst-case scenario for generalization because it partitions the most important points (i.e., highest error) into separate regression problems, which allows them to be over fit.

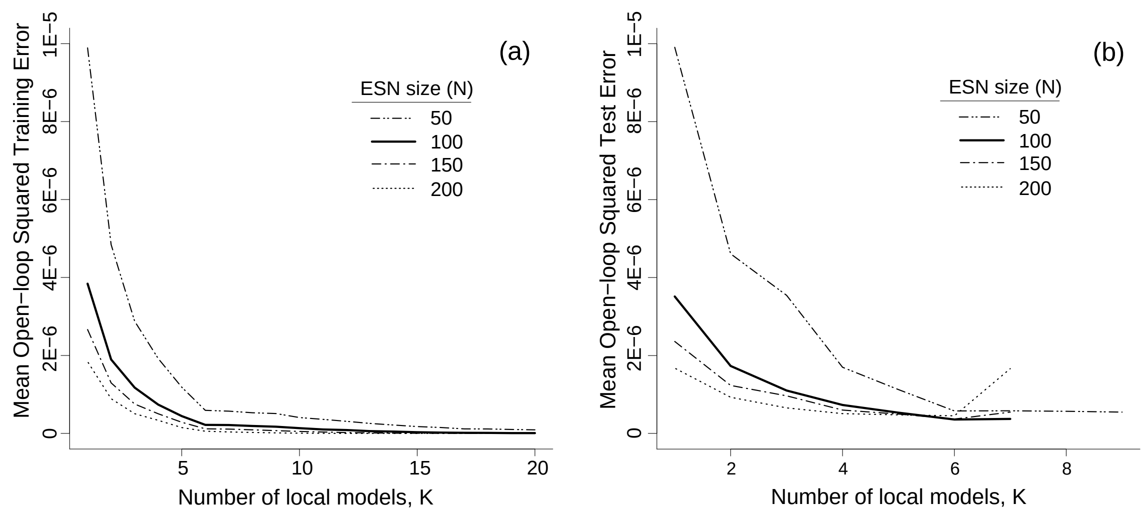

Figure 7.4: Open-loop MoR modeling of the Mackey-Glass attractor: (a) training error, (b) test error.

## 7.3 Extending MoR to Reinforcement Learning Domains

The MoR approach was originally envisioned for stationary attractors. However, there are compelling arguments that it could be useful for improving ESN modeling performance of nonstationary attractors, such as reinforcement learning domains. The first argument is already well-established, by learning examples provided in Chapters 5 and 6. A nonstationary attractor can be approximated as a series of intermediate static attractors. This notion motivates the training of the ESN readout weights via windowed linear regression. Improved modeling of the intermediate attractors would provide higher quality state-action pairs for future training windows, which could, in theory, improve performance of the ESN architecture in these domains. The second argument is based on the structure of reinforcement learning predictions compared to system prediction tasks. In reinforcement learning, unlike in system identification studies, open-loop prediction is allowable. The MoR approach is substantially more successful in improving performance in these circumstances, because step-wise state update ameliorates the accumulation of error at partition boundaries seen in the closed-loop studies of Mackey-Glass and Lorenz studies, described in previous sections. As an example, the training and testing prediction performance on the Mackey-Glass attractor is depicted in Figure 7.4.

Figure 7.4 validates previous predictions that the training prediction error approaches zero as the number of partitions approaches the number of training examples. This would make sense in that each data point would have its own model, making zero error possible. Second, Figure 7.4(b) demonstrates mostly monotonically decreasing open-loop test prediction error as the number of partitions increases. In these experiments performance increases until overfitting occurs at which point performance decreases and does not return.

The third argument is based not in improving the performance of the existing method but in facilitating higher-level abstraction into the agent—incorporation of Q-value information into the partitions. During analysis of ESN modeling of the non-Markovian state-space in Chapter 5 the factor determining sub-optimal performance was the ESN's inability to distinguish between trajectories with large differences in long term evolution. This weakness may be attributable to absence of past Q-value information being incorporated into the prediction. Incorporating past Q-value information into echo state tightly couples the Q-value predictions through time, exposing the ESN to bifurcations. Removing this feedback, while weakening the ESN's ability to predict, minimizes the architectures exposure to bifurcations. However, it seems foolish to waste this information. The MoR approach allows for the Q-value information to be incorporated into the prediction model without direct coupling between past and future values. The MoR approach would incorporate the Q-values as an additional dimension of discrimination between echo states. It seems reasonable that this technique could improve ESN modeling performance in reinforcement learning domains where high-level trajectory knowledge is useful.

# Chapter 8

# Conclusions

The goal of this research was to explore non-Markovian reinforcement learning domains as nonstationary dynamic systems. From this perspective, non-Markovian reinforcement learning transforms into the problem of modeling a domain's underlying attractor. What makes reinforcement learning a unique class of dynamic system is that the attractor underlying this system is intrinsically mobile. The learning process, itself, induces structural changes to the attractor.

Exploration of non-Markovian reinforcement learning, using this intuition, must necessarily include a study of attractor dynamics, which takes two forms. Mobility, mentioned above, describes the change in relative locations of the attractor's equilibria. Bifurcation is the other form, defined as a change in the number or stability class of the attractor's equilibria. A second, more nebulous, entity is attractor complexity, which may include the number of equilibria, but is better defined as the number and interaction of dynamic components making up the attractor.

Understanding the attractor underlying a reinforcement learning domain requires understanding how the architecture of the critic influences mobility. The locality of the model and the way in which the model observes the system influences the attractor's development. Models using local representations of the reinforcement learning domain's feature space (i.e., lookup table) trend toward large numbers of insignificant bifurcations. Global representations of the feature space trend toward small numbers of highly significant bifurcations. Adaptive features behave as a composite of both local and global representations.

Consideration of non-Markovian reinforcement learning domains, however, changes the concept of feature space representation. Without complete state, the underlying architecture is forced to represent state-space dynamically. An important notion in this research is the awareness that all feature spaces of the reinforcement learning domain rely on the underlying dynamics. The difference between Markovian and non-Markovian domains is that the Markovian domains extract system dynamics into the state-space. An example of this is the incorporation of velocity, a time derivative of position, into the state-space of Mountain Car. Takens Theorem, however, signals the ability, either explicitly or implicitly, to construct projections between state-space representations. Temporal structure of trajectories within state-space may be built into approximations of time-derivative information, which facilitates the learning of non-Markovian domains. All of this discourse must always be closed, however, with the knowledge that all state-space representations, Markovian or non-Markovian, map onto the system's dynamic feature space, the intrinsic feature space making up the system's attractor.

Application of ESN to modeling non-Markovian reinforcement learning domains validated, experimentally, the architecture's positive performance and dynamic attributes. ESN exhibits low-mobility learning through trajectory-based features. ESN performance on **non-Markovian** domains is similar to Markovian learning with traditional non-local methods *if* ESN dynamics well match the dynamics of the problem domain. ESN performance on the non-Markovian Acrobot problem is competitive with Markovian CMAC's results using only 2.3% the number of trainable parameters.

Limitations of the ESN approach to non-Markovian reinforcement learning were discovered when applying it to model more complex domains—the Modified Mountain Car Problem (MMCP) and Acrobat. ESN models of the MMCP domain exhibited either excellent modeling of the attractor or no learning at all. Through analysis of the attractor underlying this domain, the cause of this performance discrepancy was determined to be a lack of reservoir richness. The attractor underlying the learned MMCP domain exhibits two dynamic components, an unstable spiral surrounding the start state and a limit cycle encircling zero velocity states in the goal region. Moreover, modeling of these components existed at two levels of granularity. The unstable spiral could be modeled with gross accuracy where multiple policies were suitable to enter the goal region. Within the goal region, however, fine-grain action selection was necessary to ensure that the agent remained within the goal region long enough to achieve the goal. The ESN, while excellent for modeling the unstable spiral structure in both the MCP and MMCP domains, did not exhibit sufficient richness to consistently represent both scales of the MMCP.

The issue of scaling is better supported by examining the excellent results achieved when modeling the Acrobot domain via ESN. While higher dimensional, four rather than two dimensions, ESN performance was competitive with the CMACs approximation on this domain [72], despite the ESN observing only partial state information. ESN performance was consistent throughout all learning trials, indicating that the ESN exhibited reservoir richness sufficient to the model the Acrobot. Combined with knowledge gained from the MCP and MMCP experiments, it seems reasonable to acknowledge that complexity itself does not negatively influence the ESN. Rather, the ESN architecture exhibits difficulty in generating, through normal parameter selection, sufficient temporal diversity in the echo states to model problems having multiple time-scales.

A secondary issue, evident in both MMCP and Acrobot, is the ESN's inability to build intermediate state representations without consistent differentiation in the reward signals. Unless goal states are regularly achieved, the ESN is unable to achieve complex goal states when learning under high probability of exploration. This suggests a different training strategy is needed for the ESN in domains where the attractor is nonstationary and the goal state requires recognition of long term temporal dependencies. One such strategy, the use of reward shaping, sufficed in achieving sufficient intermediate goal states for the ESN to learn complex policies.

From these results a number of conclusions concerning ESN performance and utility as a non-Markovian reinforcement learning architecture may be drawn. Complex short term tasks and uniformly structured intermediate length tasks are amenable to non-Markovian representation using the ESN architecture. The technique is robust and easily trained once a reservoir with suitable dynamics is found. Reservoir designers should take note that construction of a reservoir must take into consideration both the dynamic structure of the target system as well as the structure of the reward signals. For example, in minimum path problems such as those described in this work, the reward signals form monotonically increasing Q-value trajectories, which are best modeled by asymmetric echoes. To resolve this issue, the square of the echo state was concatenated to the original echo state to ensure the existence of both symmetric and asymmetric temporal features in the echo space.

A drawback of the ESN architecture made evident by this work is the ESN's lack of memory consolidation over long time periods. The dynamic memory of the ESN, while highly capable of modeling short, complex tasks, cannot build longer term dynamics through adaptation. The memory length of the ESN has been linked to the size of the reservoir [31]. Yet even using large ESNs, noise within the reservoir, originating either from unrepresentative dynamic components or past random actions injected into the dynamic memory, make recognition of long term dependencies difficult when modeling non-Markovian reinforcement learning domains. A lesson drawn from this research is that successful use of ESN on difficult control problems requires a mechanism for consolidating and abstracting past experiences into a high-level, long term memory structure. This type of memory would require working at multiple levels of both time and space. This extension is necessary to accurately control the next level of complex, multiscale, high-dimensional systems, for example, solving the Acrobot full inversion and balance problem in a partially observable state-space.

Advances in hierarchical ESN training already exist for stationary attractor modeling. In Chapter 7 the Mixture of Readouts (MoR) approach was introduced. MoR supplies the means for increasing ESN flexibility, allowing it to model complex attractor dynamics with small reservoir size. Consequently, the MoR approach could be used to model attractor granularities existing on multiple scales, similar to those exhibited by the swing-up and balance class of control problems. Using high-level, unsupervised models to partition the attractor into simple dynamic components is a step toward consolidating memory and abstracting this memory into a high-level transition function. MoR shows suitability for use in the reinforcement learning domain due to it's excellent scale-up on open-loop predictions of test data. This class of modeling problem is very similar to the assumptions made in reinforcement learning where state update occurs at every time-step.

## Chapter 9

# Future Work

This research has provided groundwork for applying the ESN architecture to non-Markovian reinforcement learning domains. The control problems used as case studies here, Mountain Car, Modified Mountain Car, and Acrobot are just a first step. The full potential of the ESN has not yet been unlocked. Reservoirs are robust, stable temporal embeddings with well-known mathematical properties. Yet the reservoir architecture is still in its infancy. Little theory exists to motivate how to construct an appropriate reservoir for the problem at hand and only recently has research touched on methods for adapting reservoir architectures to specific domains, particularly methods that maintain the ESNs most beneficial properties—guaranteed stability, robustness, and training via linear regression.

Based on the non-Markovian reinforcement learning case studies in this work, the most pressing technical challenge to be overcome before ESNs scale to much harder domains is that of memory consolidation and state abstraction in the echo states. The current reservoir scheme, while excellent for simple, temporally short term problems is not well-suited to scales and temporal dependency lengths of real-world problems. Unfortunately, real-world domains are where non-Markovian representations would be most useful. To solve these types of problems, the reservoir's low-level advantages must be assisted by high-level memory management that abstracts both echo states and experiences into a static framework spanning long time dependencies. Mixture of Readouts, described in Chapter 7, is one approach for extending ESNs in this way. The *mixture model* subsumes the duties of abstracting and remembering important high-level dependencies over long periods of time. No specific mechanism has been proposed for this model, but as was demonstrated in Chapter 7, simple, unsupervised clustering techniques show promise and should be pursued. These models maintain all of the beneficial properties of the ESN while potentially eliminating the most significant weakness.

The need for memory and abstraction in the ESN framework has been pursued by other researchers, notably Jaeger's hierarchical ESN framework [34] and Steil's plasticity learning rules for ESN [70]. Of these, Jaeger's framework offers the most natural means of scaling the ESN to hard problems because it focuses on abstracting echoes into more general features. This framework also works well for step-wise update, which fits into the canonical TD-error update scheme of reinforcement learning. Unclear is how these techniques will perform when applied to nonstationary attractors. Finding ways to preserve old states and experiences, which may contain important information toward solving the problem optimally, while applying current abstraction to traverse the attractor efficiently is an important problem, and is a prime area of interest for further research in non-Markovian reinforcement learning domains.

Another potentially exciting direction that parallels current reinforcement learning research is an extension of the reservoir construction technique preliminarily proposed by Ozturk and Principe [56]. This technique allows knowledge of the state-space to be incorporated directly into the construction of the reservoir. The attractor structure could be sampled and then used to construct a reservoir amenable to the problem at hand. However, dynamic considerations must be made when working in non-Markovian reinforcement learning domains. The theory of this technique considers only stationary attractors where a sampling of the state-space corresponds to the underlying attractor of interest. In reinforcement learning the attractor is mobile. This technique would have to be modified such that reservoir construction scheme occurs iteratively, which could account for attractor mobility. Along these lines, there is some evidence from this work that reservoirs composed of prefabricated control elements well-suited to common attractor structures would be beneficial in adaptive control—reservoirs that form spirals and limit cycles would be very beneficial for linearly mapping attractors underlying many reinforcement learning domains.

In parallel with technical advancement to be made in the architectures for modeling reinforcement learning domains, this work has highlighted a potential theoretical advancement that may be of some consequence in future research. The attractor underlying the reinforcement learning domain is a fundamental representation of the interaction of the agent and system. The fixed point structure of this attractor provides information on the long term behavior of trajectories starting from any point within the observed state space. Therefore, it may be possible to use the attractor's structure as a means of measuring how well a problem is satisficed. That is, using the attractor's structure as dynamic partitions of the problem, it may be possible to determine the degree to which the problem has been solved in terms of all possible initial states. For example, if a stable spiral or limit cycle is known to exist within the attractor, and the goal state does not fall within these structures then it should be possible to prove that the problem has not been satisfied for states falling within the boundaries of these structures. This leads to the problem of finding attractors that satisfice the learning domain—attractors which contain no dynamic components that isolate portions of the initial state-space. Research into this type of reinforcement learning could develop an entire new class of reinforcement learning algorithms, which first attempt to learn a satisficed representation of the problem, and only then, attempt to manipulate the attractor's structure such that the problem is optimized.

# REFERENCES

[1] Shun-ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

[2] Shun-ichi Amari. Singularities affect dynamics of learning in neuromanifolds. *Neural Computation*, 18:1007–1065, 2006.

[3] Peter J. Angeline, Gregory M. Saunders, and Jordan P. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(1):54–65, January 1994.

[4] Amira F. Atiya. New results on recurrent network training: Unifying the algorithms and accelerating convergence. *IEEE Transactions on Neural Networks*, 11:3:697–709, 2000.

[5] S. Bengio, F. Fessant, and D. Collobert. Use of modular architectures for time-series prediction. *Neural Processing Letters*, 3(2):101–106, 1996.

[6] Nils Bertschinger and Thomas Natschlager. Real-time computation at the edge of chaos in recurrent neural networks. *Neural Compuation*, 16(7):1413–1436, 2004.

[7] Alejandro Bia. A study of possible improvements to the Alopex training algorithm. *Brazilian Symposium on Neural Networks (SBRN)*, 00:125–130, 2000.

[8] Alejandro Bia. Alopex-B: A new, simpler, but yet faster version of the Alopex training algorithm. *International Journal of Neural Systems*, 11(6):497–507, 2001.

[9] Gary Boone. Efficient reinforcement learning: Model-based acrobot control. In *ICRA 97*, 1997.

[10] Wray L. Buntine and Andreas S. Weigend. Computing second derivatives in feed-forward networks: A review. *IEEE Transactions on Neural Networks (T-NN)*, pages 480–488, 1993.

[11] Keith Bush and Batsukh Tsendjav. Improving the richness of echo state features using next ascent local search. In *Proceedings of the Artificial Neural Networks in Engineering Conference*, St. Louis, MO, 2005.

[12] Keith A. Bush and Charles W. Anderson. Modeling reward functions for incomplete state representations via echo state networks. In *Proceedings of the International Joint Conference on Neural Networks*, pages 2995–3000, Montreal, Quebec, 2005.

[13] Keith A. Bush and Charles W. Anderson. Exploiting iso-error pathways in the N,k-plane to improve echo state network performance. In *Neural Information Processing Systems 2006 Workshop on Reservoir Computing*, 2006.

[14] Zhe Chen, Simon Haykin, and Suzanna Becker. Sampling-based Alopex algorithms for neural networks and optimization. Technical report, Adaptive Systems Lab, McMaster University, 2003.

[15] David J. Christini, James J. Collins, and Paul S. Linsay. Experimental control of high-dimensional choas: The driven double pendulum. *Physical Review E*, 54:4824–4827, 1996.

[16] Arthur Dempster, Nan Laird, and Donald Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[17] Wlodzislaw Duch. Alternatives to gradient-based neural training. In *Fourth Conference on Neural Networks and their Applications*, pages 59–64, 1999.

[18] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[19] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 524–532, Denver 1989, 1990. Morgan Kaufmann, San Mateo.

[20] L. Feldkamp, D. Prokhorov, C. Eagan, and F. Yuan. *Nonlinear modeling: Advanced blackbox techniques*, chapter Enhanced Multi-Stream Kalman Filter Training for Recurrent Networks, pages 29–53. Kluwer Academic Publishers, 1998. Editor J. Suykens and J. Vandewalle.

[21] Robert M. French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3:128–135, 1999.

[22] Fredrick Garcia and Seydina M. Ndiaye. A learning rate analysis of reinforcent learning algorithms in finite-horizon. In *Proc. 15th International Conf. on Machine Learning*, pages 215–223. Morgain Kaufmann, San Francisco, CA, 1998.

[23] Robert Haschke and Jochen J. Steil. Input space bifurcations of recurrent neural network. *Neurocomputing (ESANN 2004 Special Issue)*, 64C:23–38, 2005.

[24] Simon Haykin. *Neural networks: A comprehensive foundation.* MacMillian College Publishing Company, New York, NY, 1994.

[25] Simon Haykin, Zhe Chen, and Suzanna Becker. Stochastic correlative learning algorithms. *IEEE Transactions on Signal Processing*, 52(8):2200–2209, 2004.

[26] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 6(2):107–116, 1998.

[27] Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

[28] Robert A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1:295–307, 1988.

[29] Robert A. Jacobs and Michael I. Jordan. A competitive modular connectionist architecture. In *Advances in Neural Information Processing Systems*, pages 767–773, 1990.

[30] Herbert Jaeger. The echo state approach to analysing and training recurrent neural networks. Technical Report GMD Report 148, German National Research Center for Information Technology, St Augustine, Germany, 2001.

[31] Herbert Jaeger. Short term memory in echo state networks. Technical Report GMD Report 152, German National Research Center for Information Technology, Saint Augustine, Germany, 2002.

[32] Herbert Jaeger. Tutorial on training recurrent neural networks covering BPTT, RTRL, EKF and the echo state network approach. Technical Report GMD Report 159, German National Research Center for Information Technology, Saint Augustine, Germany, 2002.

[33] Herbert Jaeger. Adaptive nonlinear system identification with echo state networks. In *Advances in Neural Information Processing Systems*, volume 15, pages 593–600. MIT Press, 2003.

[34] Herbert Jaeger. Discovering multiscale dynamical features with hierarchical Echo State Networks. Technical Report 9, Jacobs University, Bremen, Germany, 2007.

[35] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304:78–80, 2004.

[36] Herbert Jaeger, Mantas Lukosevicius, and Dan Popovici. Optimization and applications of echo state networks with leaky integrator neurons. *Neural Networks*, 20(3):335—352, 2007.

[37] M. I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural Computation*, 6:181–214, 1994.

[38] Michael Jordan. Serial order: A parallel distributed processing approach. Technical report, Institute for Cognitive Science, University of California, San Diego, 1986.

[39] Michael I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In Erlbaum, editor, *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 531–546, Hillsdale, NJ, 1986.

[40] Leslie Pack Kaebling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237—285, 1996.

[41] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82:35–45, 1960.

[42] Timo Koskela. *Neural network methods and modelling time varying processes.* PhD thesis, Helsinki University of Technology, 2003.

[43] R. M. Kretchmar. *A synthesis of reinforcment learning and robust control theory.* PhD thesis, Colorado State University, Fort Collins, CO, USA, 2000.

[44] R. M. Kretchmar and C. W. Anderson. Comparison of cmacs and radial basis functions for local function approximators in reinforcement learning. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 834–837, 1997.

[45] Y. LeCun, J. Denker, S. Solla, R. E. Howard, and L. D. Jackel. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, San Mateo, CA, 1990. Morgan Kauffman.

[46] Lennart Ljung. *System Identification - Theory For the User, 2nd ed.* PTR Prentice Hall, Upper Saddle River, N.J., 1999.

[47] Wolfgang Maass, Thomas Natschlager, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14:2531–2560, 2002.

[48] M. McCloskey and N. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *The Psychology of Learning and Motivation*, 24:109–164, 1989.

[49] E. Mizutani and S. Dreyfus. Totally model-free reinforcement learning by actor-critic Elman networks in non-Markovian domains. In *In Proceedings of the IEEE-INNS International Joint Conference on Neural Networks*, Alaska,USA, 1998.

[50] E. Mizutani and S. Dreyfus. On using discretized Cohen-Grossberg node dynamics for model-free actor-critic neural learning in non-Markovian domains. In *In Proceedings of the IEEE International Int'l Symposium on Computational Intelligence in Robotics and Automation*, Kobe,Japan, 2003.

[51] E. Mizutani and S. Dreyfus. Two stochastic dynamic programming problems by model-free actor-critic recurrent-network learning in non-Markovian settings. In *In Proceedings of the IEEE-INNS International Joint Conference on Neural Networks*, Budapest,Hungary, 2004.

[52] Martin Moller. *Efficient training of feed-forward neural networks*. PhD thesis, Computer Science Department, Aarhus University, Aarhus, Denmark, 1997.

[53] M. C. Mozer. *Neural net architectures for temporal sequences processing*, volume 15, pages 243–264. Addison Wesley, Reading, MA, 1993. A. S. Weigend and N. A. Gershenfeld.

[54] Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: theory and application to reward shaping. In *Proc. 16th International Conf. on Machine Learning*, pages 278–287, 1999.

[55] Mustafa C. Ozturk and Jose C. Principe. Computing with transiently stable states. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1467–1471, 2005.

[56] Mustafa C. Ozturk, Dongming Xu, and Jose C. Principe. Analysis and design of echo state networks. *Neural Computation*, 19:111–138, 2006.

[57] Barak A. Pearlmutter. Gradient calculations for dynamic recurrent neural networks: A survey. *IEEE Transactions on Neural Networks*, 6(5):1212–1228, September 1995.

[58] Morten With Pedersen. *Optimization of recurrent neural networks for time series modeling*. PhD thesis, Technical University of Denmark, Copenhagen, Denmark, 1997.

[59] P. Ploger, A. Arghir, T. Gunther, and R. Hosseiny. Echo state networks for mobile robot modeling and control. In *In Proceedings of the ROBOCUP 2003*, 2003.

[60] J. Principe, N. Euliano, and W. Lefebvre. *Neural and adaptive systems*. John Wiley and Sons, New York, NY, 2000.

[61] Danil Prokhorov. Echo state networks: Appeal and challenges. In *Proceedings of International Joint Conference on Neural Networks*, pages 1463–1466, 2005.

[62] Larry D. Pyeatt and Adele E. Howe. Decision tree function approximation in reinforcement learning. Technical Report TR CS-98-112, Colorado State University, Fort Collins, Colorado, 1998.

[63] Y. N. Rao, S.-P. Kim, J. C. Sanchez, D. Erdogmus, J.C. Principe, J.C. Carmena, M.A. Lebedev, and M.A.L. Nicolelis. Learning mappings in brain-machine interfaces with

echo state networks. In *IEEE International Conference on Acoustic, Speech, and Sig. Proc., Philadelphia, PA, USA, March 18-23*, 2005.

[64] R. Ratcliff. Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychological Review*, 97:285–308, 1990.

[65] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[66] J. W. Sammon. A non-linear mapping for data structure analysis. *IEEE Transactions on Computers*, C-18:401–409, 1969.

[67] J. D. Schaffer, D. Whitley, and L. J. Eshelman. Combinations of genetic algorithms and neural networks: A survey of the state of the art. In D. Whitley and J. D. Schaffer, editors, *Proc. Int. Workshop Combinations Genetic Algorithms Neural Networks (COGANN-92)*, pages 1–37, Los Alamitos, CA, 1992. IEEE Comput. Soc. Press.

[68] Jochen J. Steil. Backpropagation-decorrelation: Online recurrent learning with O(N) complexity. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 843–848, 2004.

[69] Jochen J. Steil. Online stability of backpropagation-decorrelation recurrent learning. *Neurocomputing*, 69(7-9):642–650, 2006.

[70] Jochen J. Steil. Online reservoir adaptation by intrinsic plasticity for backpropagation-decorrelation and echo state learning. *Neural Networks*, 20:353–364, 2007.

[71] Steven H. Strogatz. *Nonlinear Dynamics and Chaos.* Westview, 1994.

[72] Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems*, volume 8, pages 1038–1044. MIT Press, 1996.

[73] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction.* The MIT Press, Cambridge, MA, 1998.

[74] Floris Takens. Detecting strange attractors in turbulence, dynamical systems and turbulence. In D. A. Rand & L. S. Young, editor, *Lecture Notes in Mathematics*, volume 898, pages 366–381. Springer, 1981.

[75] K.P. Unnikrishnan and K.P. Venugopal. Alopex: A correlation-based learning algorithm for feedforward and recurrent neural networks. *Neural Computation*, 6:469–490, 1994.

[76] Tijn van der Zant, Vlatko Becanovic, Kazuo Ishii, Hans-Ulrich Kobialka, and Paul Ploeger. Finding good echo state networks to control an underwater robot using evolutionary computations. In *A proceedings volume from the 5th IFAC Symposium*, 2004.

[77] P. Werbos. Backpropagation through time: What it does and how to do it. *In Proceeding of the IEEE*, 78:1550–1560, 1990.

[78] Darrell Whitley. Genetic algorithms and neural networks. *Genetic Algorithms in Engineering and Computer Science*, Winter, Periaux, Galan and Cuesta, eds.:203–216, John Wiley, 1995.

[79] Alexis P. Wieland. Evolving controls for unstable systems. In *In Connectionist Models: Proceedings of the 1990 Summer School*, Pittsburgh,USA, 1990.

[80] Alexis P. Wieland. Evolving neural network controllers for unstable systems. In *In Proceedings of the IEEE-INNS International Joint Conference on Neural Networks*, Seattle,USA, 1991.

[81] R. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280, 1989.

[82] Ronald J. Williams. Some observations on the use of the extended Kalman filter as a recurrent network learning algorithm. Technical Report NU-CCS-92-1, Northeastern University, College of Computer Science, Boston, Massachusetts, 1992.

[83] Ronald J. Williams. Training recurrent networks using the extended Kalman filter. In *Proceeding of the International Joint Conference on Neural Networks*, volume IV, pages 241–246, 1992.

# Appendix A

# Reinforcement Learning Algorithms

```
[state, oldstate, oldaction] ← initialize();
Q ← **0**;
oldQ ← Q;
firsttime ← TRUE;
while !converge(Q, oldQ, threshold) do
    action ← argmax[actionset](Q(state, action));
    if firsttime then
        firsttime ← FALSE;
    else
        if isgoal(state) then
            Q ← update(Q(state, action), reward ← 0);
            [state, oldstate, oldaction] ← initialize();
        end
        Q ← update(Q(oldstate, oldaction), reward ← -1);
    end
    oldstate ← state;
    oldaction ← action;
    oldQ ← Q;
    state ← nextstate(state, action);
end
```

**Algorithm 1**: Pseudocode for SARSA training of the lookup table.

```
NN ← 0;
S ← learningschedule();
M ← maxstepschedule();
firsttime ← FALSE;
for s ← 1 to < |S| do
    for 1 to R do
        [state, oldstate, oldaction] ← initialize();
        firsttime ← TRUE;
        for 1 to M[s] do
            if rand() < S[s] then
                randaction ← TRUE;
                action ← selectrandaction[actionset]();
            else
                randact ← FALSE;
                action ← argmax[actionset](predictQ(NN, state, action));
            end
            if firsttime then
                firsttime ← FALSE;
            else
                if goal(state) then
                    if !randaction then
                        NN ← updateNN(NN, state, action, reward ← 0);
                    end
                    [state, oldstate, oldaction] ← initialize();
                end
                if !randaction then
                    NN ← updateNN(NN, oldstate, oldaction, reward ← -1);
                end
            end
            oldstate ← state;
            oldaction ← action;
            state ← nextstate(state, action);
        end
    end
end
```

**Algorithm 2**: Pseudocode for SARSA training of the feedforward neural network.

updateNN(NN,state,action,reward): returns the NN after updating the hidden and out weights via backpropagation of errors.

```
Q ← **0**;
S ← learningschedule();
M ← maxstepschedule();
ESN ← initializeESN();
firsttime ← FALSE;
for $s \leftarrow 1$ to $< |S|$ do
    for 1 to R do
        [state, oldstate, oldaction] ← initialize();
        ESN ← resetESN(ESN);
        firsttime ← TRUE;
        for $t \leftarrow 1$ to M[s] do
            if rand() < S[s] then
                randaction ← TRUE;
                action ← selectrandaction[actionset]();
            else
                randact ← FALSE;
                action ← argmax[actionset](predictQ(ESN, nonMarkov(state),
                action));
            end
            ESN.reservoir ← iterateESN(ESN, nonMarkov(state), action);
            if firsttime then
                firsttime ← FALSE;
            else
                if goal(state) then
                    reward ← 0;
                    [state, oldstate, oldaction] ← initialize();
                    ESN ← resetESN(ESN);
                else
                    reward ← -1;
                end
            end
            echolist ← {echolist, ESN.z};
            rewardlist ← {rewardlist, reward};
            randactionlist ← {randactionlist, randaction};
            oldstate ← state;
            oldaction ← action;
            state ← nextstate(state,action);
            if mod(t,lrwindow)=0 then
                Qtarget ← rowSums(embed(rewarlist, horizon));
                solveindices ← which(randactionlist==FALSE);
                ESN.readout ← solve(echolist[solveindices], Qtarget[solveindices]);
            end
        end
    end
end
```

**Algorithm 3**: Pseudocode for SARSA training of the Echo State Network.

nonMarkov(state): returns only the observable components of the complete, Markovian, state.

resetESN(ESN): returns the ESN after assigning both the internal echo state, $\mathbf{z}$ and the ESN.readout weights to $\mathbf{0}$.

iterateESN(ESN): returns the ESN after updating the internal echo state, $\mathbf{z}$, according to Equation 3.9.

solve(echolist[solveindices],Qtarget[solveindices]): returns the set of weights after solving the system given in Equation 3.11.