Abstract—The Echo State Network (ESN) architecture is used as a recurrent network strategy for approximating the Q-function of the mass-spring-damper linear dynamical system when only partial state is observable. The ESN architecture’s approximation performance is compared against feed-forward neural networks given perfect state information (FNN) and a finite window of time-delayed partial state (TDNN), respectively. Both feed-forward representations are known to perform well in approximating the Q-function in this problem domain. We demonstrate that the ESN, given partial state, well-represents temporally dependent rewards and exhibits similar performance to FNN and TDNN architectures in approximating the Q-function during on-line learning.

I. INTRODUCTION

In the physical world, time is the medium of change. Recognizing temporal structure, therefore, is a core learning task in the physical world. A common challenge faced in modern control research is learning to interact with dynamical systems. Discrete dynamical systems are an abstraction of the physical world in which time has been discretized to arbitrary coarseness, $\Delta t$, and the world is represented as a state vector $x_t$, having some linear, nonlinear, stochastic, or hybrid mapping to state $x_{t+\Delta t}$. Computer science and engineering research is driven by the desire to enable machines to interact with the physical world, explore its state-space, and learn its hidden structure independently of human guidance.

Fundamental to realizing this ultimate goal are two technologies. First, there must exist a formal, high-level learning strategy for assessing the quality of any action, given any state of the world, and set of goals to achieve. This assessment process defines the optimal policy. Given a perfect assessment of the quality of any action in a given state, the optimal policy is the action yielding the highest quality for that state with respect to a desired goal. Second, there must exist an effective, low-level technique for learning the relationships between state, action, and quality given that changes in quality may contain hidden dependencies connected through both state-space and time. This has been the subject of research in the field of reinforcement learning (RL) [1].

Recurrent neural networks (RNN) are one approach to discovering nonlinear dependencies in time. Mizutani and Dreyfus have proposed the use of Elman networks for learning state quality given incomplete state information [2], [3] for small, longest-path problems. In dynamical systems control, Wieland [4] demonstrated evolutionary search mechanisms for solving this class of problem. A number of other techniques exist for training recurrent neural networks [5], [6].

This paper investigates an Echo State Network (ESN) [7], [8] architecture as the approximation of the Q-function for temporally dependent rewards embedded in a linear dynamical system, the mass-spring-damper (MSD). This problem has been solved utilizing feed-forward neural networks (FNN) when all state information necessary to specify the dynamics is provided as input [9]. Time-delayed neural networks (TDNN) solve this problem with finite-size windows of incomplete state information. Our research demonstrates that the ESN architecture represents the Q-function of the MSD system given incomplete state information as well as current feed forward neural networks given either perfect state or a temporally-windowed, incomplete state vector.

The remainder of this paper is organized as follows. We introduce basic concepts of reinforcement learning and the Echo State Network architecture in Sections II and III, respectively. The MSD system simulation is defined in Section IV. Experimental results for learning state quality given incomplete state information are presented in Section V. Results for learning estimates of all future state qualities for incomplete state information is presented in Section VI. Section VII discusses the potential of the ESN for use in reinforcement learning and provides current and future directions of research.

II. REINFORCEMENT LEARNING: Q-LEARNING

To understand how low-level temporal learning can empower reinforcement learning techniques, we introduce the framework of a popular RL implementation, Q-learning [1]. Assume a system of states, $s$, and actions, $a$, where $a$ is an operator defining change of state.

Define time, $t$, and specify that state $s_{t+1}$ results from taking action $a_t$ from state $s_t$, yielding reward $r_{t+1}$. Q-Learning may be characterized as the learning of an approximate function, $Q(s_t, a_t)$, for an optimal action-value function, $Q^*(s_t, a_t)$. The optimal action-value, $Q^*(s_t, a_t)$, is the maximum expected sum of all future rewards for taking action $a_t$ from state $s_t$. Let $Q(s_t, a_t) \approx Q^*(s_t, a_t)$ where we define:
The discount factor, \( \gamma \), is a means of bounding the value of future reward when summing over an infinite future horizon of state-action pairs.

The challenge of implementing Q-Learning resides in finding ways to represent \( Q(s_t, a_t) \). The relationship between reward \( r \) and the dynamical system comprised of states \( s \) through which the learner interacts via actions \( a \) will likely contain both long and short term temporal dependencies. Possibly, these dependencies are unobserved time derivatives of state values. This is the fundamental problem of employing RL techniques within dynamical systems and signifies the importance of developing learning techniques which effectively handle temporal dependencies. In practice, developing a representation of the approximate action-value function for non-trivial dynamical systems is a challenging problem.

### III. Echo State Networks

The Echo State Network (ESN), and related Liquid State Machine, are models of temporal learning developed concurrently by Jaeger [7], [10] and Maass [8]. Theoretically, the methods are identical, but they differ in the dynamics of the network’s activation functions. The ESN is closer to traditional RNN architectures in that it employs either sigmoid or tanh activation functions within the hidden units [11]. Here we focus on the ESN model.

The primary motivation of the ESN is to avoid the slow convergence, computational complexity, and instability of recurrent network learning algorithms [7]. ESNs rely on the inherent dynamics of a large but sparsely connected recurrent structure in which many small, loosely-coupled dynamical systems interact with the input data to build a large, rich set of features.

In our research we have modified the canonical form of an ESN [7] to allow for direct comparison with feed-forward network approaches by decoupling the Echo State Network feature generation mechanism from the learning mechanism, see Figure 1. We define an ESN as a two layer network in which there are \( K \) input nodes, and \( N \) hidden nodes. The states of these nodes are represented as column vectors \( u \) and \( s \), respectively. The system is further specified by two matrices defining the connection topology: 1) input connections onto hidden nodes, \( W_{in} \), and 2) recurrent connections of hidden nodes, \( W_{esn} \). In each of these matrices, element \( w_{ij} \) represents a connection onto node \( i \) from node \( j \). The ESN is a dynamical system. Temporal mappings of the internal state are defined by the following recurrence relation [7].

\[
{s_{t+1} = f(W_{in}u_{t+1} + W_{esn}s_t),}
\]

where activation function \( f \) is some nonlinear function such as sigmoid or tanh.

Typically, an ESN is defined in terms of an input, hidden, and output layer where the training rule over an output matrix \( W_{out} \) is considered a batch or recursive least squares fit of the output states, \( y \), as a linearly weighted sum of the ESN hidden state, \( s \), through time. The training rule is extremely efficient and may be thought analogous to solving an over-specified system of linear equations, assuming the number of training examples is large with respect to the size of the matrix. Here, we replace the linear output function with a standard feed-forward neural network trained via error-backpropagation. We define this architecture as the Echo State Neural Network (ESNN).

The power of the ESN lies in its rich feature generation capability that can be combined with a learning mechanism to map features to desired outputs. Put differently, the matrix \( W_{esn} \) defines a set of dynamical basis functions that generate features of the input which are then mapped into the output-space [8]. Moreover, the ESN architecture provides a mathematically well-defined mechanism, the Echo State Property (ESP), for scaling the weights of matrix \( W_{esn} \).

#### A. The Echo State Property

An elegant property of the ESN framework is that the weight matrix \( W_{esn} \) has a scaling rule given explicitly via the Echo State Property [7], [8]. A sketch of the formal mathematical justification, originally presented by Jaeger [7], follows.

**Definition 1:** Assume \( U \) is a closed and bounded set of inputs. The ESN has echo states if \( s \) is uniquely determined by any left-infinite input sequence, \( u_{-\infty} \).

Definition 1 is alternatively described by Maass as the *point-wise separation property* [8]. In other words, the Echo State Property exists if at any point in an infinite computational sequence, the state vector \( s \) is uniquely determined by the previous inputs (left-infinite input sequence), and therefore, permits separation of any input sequence, \( u_{-\infty} \). This is a key component of a dynamic feature generator. The internal state space, which may be considered the feature space of the ESN, is uniquely determined by the inputs. A mathematical
extension of this definition, formally outlining the structure of \( W_{esn} \) necessary to achieve the Echo State Property, is given by Proposition 1 [7].

**Proposition 1:** Assume an ESN network with unit output functions \( f = \tanh \). Define the network state update operator \( T \), such that \( s(n+h) = T(s(n), u^h) \) where \( s(n) \) is the internal state at time \( n \), \( u^h \) is a finite input sequence of length \( h \), and \( s(n+h) \) is the state resulting from iterative application of Equation 1 for this initial state and input sequence. Let the weight matrix, \( W_{esn} \), satisfy \( \sigma_{\max} = \Lambda < 1 \), where \( \sigma_{\max} \) is its largest singular value. Then \( \| T(s, u), T(s', u) \| < \| s, s' \| \) for all inputs \( u \), for all states \( s, s' \in [-1, 1]^N \). Then the ESN network has echo states for all inputs \( u \). Also, let the weight matrix have a spectral radius \( \lambda_{\text{max}} > 1 \), where \( \lambda_{\text{max}} \) is an eigenvalue of \( W_{esn} \) with the largest absolute value. The network has an asymptotically unstable null state. This implies that it has no echo states for any input set \( U \) containing \( u = 0 \) and compact state set \( S = [-1, 1]^N \).

This proposition may be framed as follows. Given a random weight matrix \( W_{esn} \), and definitions of \( \sigma_{\max} \) and \( \lambda_{\text{max}} \) from Proposition 1, then any scaling factor, \( \alpha \), applied to \( W_{esn} \), satisfying the inequality, \( \alpha < 1 / | \lambda_{\text{max}} | \), guarantees the ESN containing \( W_{esn} \) maintains the Echo State Property.

**B. ESN Parameters**

The ESN contains three parameters: \( \alpha \), \( N \), and \( \rho \)—the density of nonzero entries in \( W_{esn} \). Empirical research [7] has shown that suitable recurrent weight matrices, \( W_{esn} \), are not arbitrary. In practice, the matrix \( W_{esn} \) must be sparse, having only \( \rho = 5 \text{--} 20\% \) non-zero weight values, distributed randomly [7], [12]. The scaling parameter \( \alpha \) must be in the range defined by the Proposition 1. However, the suitable scaling of \( \alpha \) within this range is an open problem as is the size of \( N \) necessary to guarantee a set of features rich enough to specify the dynamics of a given class of systems.

**IV. MASS-SPRING-DAMPER SYSTEM**

Our experimental dynamical system is the mass-spring-damper (MSD). The MSD may be defined as a theoretical spring attached to a mass, \( m \), where \( m \) is anchored at the origin, \( x = 0 \). The spring has no residual length when unstretched. The spring imparts a force, \( F_k = -kx \), onto the mass that scales linearly with respect to deflection of the mass from the origin. In addition, the MSD also incorporates a theoretical damper, \( b \), which imparts a force \( F_b = -bx \). We also define external action, \( a \), that is applied to the mass. The total force acting on the system is \( F_{\text{tot}} = F_k + F_b + a \). We defined the state of the MSD at time \( t \) to be \( s_{\text{msd}}^t = [x_t \ x_t'] \).

The MSD system is 2nd order, and we compute the next state, \( s_{\text{msd}}^{t+1} \) by numerical integration via a fourth order Runge-Kutta method over the differential equation:

\[
\ddot{x} = \frac{-kx - bx + a}{m}.
\]

For all experiments \( k = 10.0 \) (N/m), \( b = 0.07 \) (Ns/m), and \( m = 1.0 \) (kg). The integration timestep is \( \Delta t = 0.25 \) (sec). These parameters were chosen to define an underdamped system; the system is critically damped for \( b_{\text{crit}} = 6.66 \) (Ns/m).

**V. PREDICTING FUTURE REWARD**

The primary motivation of this research is to define a learning architecture that can represent temporally dependent rewards for use in RL. For our MSD simulation, the reward structure is defined as:

\[
r_{t+1} = -|x_{t+1}^s - x_{t+1}^{ret} - 0.1|\dot{x}_{t+1}. \tag{3}
\]

This structure is used for two reasons. First, the form of the positional reward term allows for future inclusion of setpoint modification where \( x_{t+1}^{ret} \) is some reference position for assigning rewards. This modification has obvious advantages when utilizing RL in the sense of dynamical systems control. For the current experimental setup, however, \( x_{t+1}^{ret} = 0.0 \) for all \( t \). The velocity term of the reward function penalizes MSD states in amounts proportional to the magnitude of velocity. This term also facilitates smooth transitions between setpoints should \( x_{t+1}^{ret} \) be varied during learning. Three learning architectures are compared on this problem.

The FNN architecture requires the complete state of the dynamical system as input, \( I_{fnn} = [x_t \ x_t'] \). The TDNN architecture, when complete state information (velocity in this case) is unobservable, uses a windowing of the partial state information such that \( I_{tdnn} = [x_t \ x_{t-1} \ \ldots \ x_{t-k}] \). The windowed input data can be considered discrete time-derivative information over the window size, \( k \). Clearly, the choice of \( k \) varies in length based on the dynamics of the system. A significant problem in the employment of these methods is that state is often incompletely observable and the window size \( k \) is often difficult to determine a priori for an arbitrary system.

The ESNN architecture alleviates the two primary concerns of FNN and TDNN methods. ESNNs represent missing state information internally. In the context of RL, we desire to show that the ESNN, given input of \( x_t \), is capable of predicting \( r_{t+1} \), and therefore, internally learns the required missing state information of the MSD system.

In our first experiment, we define three neural network architectures constructed from 20 hidden nodes and one output node. The input nodes vary according to the defined input vector for each architecture. The FNN and TDNN architectures receive the inputs \( I_{fnn} \) and \( I_{tdnn} \), described above, where the window size of \( I_{tdnn} \) was \( k = 5 \). The ESNN architecture receives input \( I_{esnn} = [x_t \ s] \) where \( s \) is the internal state of an ESN having \( N = 10 \) hidden nodes, density, \( \rho = 0.10 \), and initial weights randomly sampled from a uniform distribution \( iw = [-1.0, 1.0] \). The initial weights of the FNN, TDNN, and ESNN learning components were each randomly sampled from a uniform distribution on the range \([-1.0, 1.0] \). Learning rates for the hidden layer, \( r_{hid} \) and output layer, \( r_{out} \) were 0.5 and 0.01, respectively, for all three architectures, and were trained via on-line error backpropagation. All hidden units used \( \tanh \) activation functions. All output units were linear.
These three architectures were trained on the MSD system for 50,000 simulation timesteps (12,500 sec of simulation time). The initial MSD states, \( x_0 \) (m) and \( \dot{x}_0 \) (m/s), were randomly drawn from a uniform distribution on the range \([-1.0, 1.0]\). At each timestep an action, drawn randomly from the set \([-1.0, 1.0]\), was applied to the MSD.

Prediction error for each network at each timestep was defined as temporal difference error, \( E_{td} \), defined as:

\[
E_{td} = r_{t+1} + \gamma \hat{Q}_{t+1} - \hat{Q}_t.
\]

The initial results reported here are for \( x_t > 0 \), so each network is trained to predict only the next reward, \( r_{t+1} \). Experiments for \( \gamma > 0 \) are presented in Section VI.

We must also specify a method of comparison. We cannot know the predicted reward for an arbitrary position, \( x_t \), because predicted reward is a function of the entire history of positions visited and random actions taken. Therefore, we cannot directly generate the approximated ESNN reward surface as a function of \( x_t \) and \( \dot{x}_t \). To provide a meaningful experimental comparison, therefore, we must sample the predicted rewards and corresponding state values as training proceeds on-line. We may then compare the sampled reward surfaces with the reward surface generated directly by the FNN architecture to ensure that predictions of future reward are correct. Results of this comparison of reward prediction performance for the FNN and ESNN architectures are presented in Figure 2.

Figures 2a and b present surfaces for the predicted values of \( r_{t+1} \) sampled on-line from ESNN and FNN with respect to the set of actions, \( a = [1, 0, -1] \). Figure 2c depicts the predicted \( r_{t+1} \) surface generated from the FNN by enumeration of the state space over the set of actions.

The range of \( x \) and \( \dot{x} \) values sampled in Figure 2 is a subset of the total range of states visited by the MSD during simulation. Due to the infrequency of certain states being visited by the MSD during an actual simulation, the range was constrained to a set of states having high probability of being sampled. This constraint eliminated empty positions in the sampled surface, making comparison more clear.

Visual inspection concludes that the ESNN architecture is successfully predicting \( r_{t+1} \). The general elliptical properties of the enumerated FNN surface are preserved. Moreover, we can validate directly the surface of the FNN exact solution by direct mathematical comparison of the contour intervals with values expected from Equation 3. The most problematic feature of the ESNN is noise in the predicted rewards. This issue will be addressed again in Section VI. The sampled \( r_{t+1} \) for the TDNN is not shown in Figure 2, but visually resembled Figures 2a and b.

VI. MODELING THE Q-VALUE FUNCTION

To extend our investigation of the ESNN architecture’s performance to approximating the Q-value function, we modify our experiments such that our temporal difference error, given in Equation 4, utilizes a discount factor of \( \gamma = 0.99 \). This imposes exponential decay on each successive reward in the infinite sum of expected future rewards. In addition, we wish to demonstrate the stability of the ESNN’s learned Q-value function compared with FNN and TDNN as action selection becomes deterministic during on-line learning.

Rather than generating actions randomly, we implemented an action selector that, for a given neural architecture, computes values of \( Q_{t+1} \) for the current state and three actions, \(-1, 0, \text{ and } 1\). The action yielding maximum \( Q_{t+1} \) is selected as the action taken. All neural architectures simultaneously train on-line. The FNN architecture’s approximation was used during action selection to ensure that selected actions drive the MSD system to very high-quality states.

To induce convergence of the networks toward relatively high values of \( Q_{t+1} \), a training schedule of random exploration versus deterministic exploitation of the learned Q-function was employed. The training run was 35,000 simulation timesteps for a total of 8,750 simulated seconds. This run was subdivided into seven sequential intervals, each of 5000 steps. The probabilities of random action selection for each interval were \( 1, 0.8, 0.6, 0.4, 0.2, 0.1, \text{ and } 0.05 \). This slowly transforms action selection from stochastic to deterministic.

Figure 3 shows the relative performance of the three architectures. Figure 3a shows that the Q-values predicted by all three architectures approximately converge by about 350 seconds of simulation time. Figure 3b shows predicted Q-values throughout the entire training run. As action selection transitions from completely random to deterministic, the MSD moves toward states having consistently high Q-values. Throughout this training schedule the three architectures produce nearly the same predictions of \( Q_{t+1} \). This result indicates that prediction noise discovered in Figures 2a and b may be attributed to our sampling method of comparison and does not directly influence the prediction capability of the ESNN.

We propose that the accuracy of these predictions can be evaluated by observing the exact \( Q_{t+1} \) surface generated by the FNN over enumeration of the state space at the conclusion of learning. This surface should be indicative of the \( Q_{t+1} \) function approximation capabilities of the three architectures. This inferential comparison is imposed by the ESNN and TDNN architectures which cannot be induced to predict \( Q_{t+1} \) for an arbitrary state without complete historic context of the input sequence, a difficult reconstruction and representation problem.

Further, the sampling method utilized in Section V to compare predicted rewards does not apply due to significant changes in the observed state space imposed by the training schedule. The exact \( Q_{t+1} \) surface predicted by the FNN network for \( a = 0 \) is presented in Figure 4. Surfaces for \( a = [1, -1] \) are similar but shifted diagonally across the \( x_t \) and \( \dot{x}_t \) axes. With respect to the known reward and intuition of how the MSD dynamics respond through time, we propose that this surface is reasonable in that the known optimal state, \([x = 0, \dot{x} = 0]\) resides near the maximum of this surface. Therefore, the shape of the surface is reasonable, despite a suboptimal convergence of the FNN. Suboptimal convergence of the FNN is attributable to the structure of our training sched-
Fig. 2. Comparison of reward surfaces with respect to state for the FNN and ESNN architectures over actions, $a = [-1, 0, 1]$. (a) $r_{t+1}$ surfaces sampled on-line for ESNN predictions of $r_{t+1}$ (b) $r_{t+1}$ surfaces sampled on-line for the FNN architecture. (c) Surfaces constructed by direct function approximation of the FNN network given enumeration of the state space. Note, contour lines represent equivalent altitude among all surface plots.

ule. Based on the relative zero prediction variance between the three architectures throughout the training schedule, we propose that all three architectures are approximating similar $Q_{t+1}$ functions, and therefore, $Q$-value function approximation capability is similar.

VII. DISCUSSION

We have proposed a novel implementation for RL in which an ESN, a fixed, stochastically generated recurrent network, was used as a feature generator to approximate future rewards given incomplete state information. Our experimental environment was the mass-spring-damper (MSD) linear dynamical system.

We have shown that the ESN features generated for incomplete state enable a feed forward neural network (ESNN) to predict future rewards, $r_{t+1}$, at a level of performance roughly equivalent to that of feed forward neural networks which may observe the complete system state (FNN) or a finite window of present and past incomplete states (TDNN). We have also demonstrated that $Q_{t+1}$ predictions on the MSD system by the ESN, TDNN, and FNN architectures converge quickly and maintain convergence throughout an online training scenario where action selection transitions from completely stochastic to highly deterministic. These results suggest that ESN features facilitate good predictions of reward, a critical component of the use of RL in dynamical systems which contain hidden temporal dependencies.

We believe that the ESN framework holds advantages over other recurrent neural network learning strategies, such as
back-propagation through time, in that the temporal feature generation and feature mapping problems are naturally decoupled. This framework enables the feature identification problem to be solved independently of learning, and, therefore, holds promise for extending RL to dynamical systems of high-complexity and long temporal dependencies. Current and future work includes empirical comparisons of fixed ESN and trainable recurrent networks, tests on more complex control problems, and analysis of the dynamics of ESNs as a function of its parameters.

Fig. 3. (a) Convergence of the FNN, TDNN, and ESNN architectures through time. The three architectures typically converge in 350-400 (sec) of simulation time. (b) Converged $Q_{t+1}$ prediction of the ESN, TDNN, and FNN architectures over a seven interval training schedule. Arrows mark boundaries where the probabilities of random actions change. The probabilities of random actions were 1, 0.8, 0.6, 0.4, 0.2, 0.1, and 0.05.

Fig. 4. Exact $Q_{t+1}$ surface for action $a = 0$ generated by enumeration over the state space for FNN architecture after training. Theoretically, the maximum Q-value, $Q_{t+1} = 0$, occurs at $[x = 0, \dot{x} = 0]$. After training, the FNN predicts $Q_{t+1} = -4.7$ for a large plane of the surface, which contains the theoretical maximum location, but has not converged to the optimal Q-value.

REFERENCES