Exogenous control and dynamical reduction of echo state networks

Patrick W. Stinson and Keith A. Bush

Abstract—In this paper, we demonstrate that a Q-Learning control policy with a Growing Neural Gas state space approximation is sufficient to control echo state neural networks of arbitrary dynamical complexity in a discrete time model, given sufficient input gain. We control through a single input unit fully connected to an echo state reservoir; our influence of the system is constrained to the input only—no weights are modified after the network is initialized. Our methodology is successful for both temporal and spatial control goals. However, control of increasingly complex systems requires increasing saturation of units’ activation function non-linearities, which we achieve by increasing the input gain. We find that when subjected to the minimal gain needed for control goals, systems of varying levels of dynamical complexity are reduced to very similar levels. However, even in such reduced circumstances, our control framework is still advantageous or necessary to achieve performance above chance levels.

I. INTRODUCTION

CONTROL in non-linear systems is an important topic, as most real-world systems exhibit complex non-linear dynamics. In such an environment, a control framework that does not rely directly on the system’s underlying dynamical structure is of particular interest, as traditional analytical models may not sufficiently capture the richness of the dynamics, or observability may be limited, making an accurate formulation of the underlying dynamics very challenging. Previous work [1] has demonstrated that Q-Learning in a state space approximated by a Growing Neural Gas (GNG) algorithm [2] is sufficient to establish stable dynamics in particular chaotic non-linear systems, in which the dynamics are known a priori, such as a nine-dimensional Lorenz attractor. Echo state networks [3], or equivalently, liquid state machines [4], provide an attractive generalization of such systems, as the recurrent reservoir offers a rich basis of dynamics. Here we extend Galadeta and Dangelmayr’s work [1] and investigate whether such a control policy is successful in echo state networks whose dynamical complexity we can scale arbitrarily by increasing the size of the network [5]. Past studies establishing control of chaotic systems (e.g., [1], [6], [7], [8]) have done so by making small perturbations to the intrinsic dynamics of the system itself. Here, we do not modify (or know) the dynamics of the reservoir; rather, we establish control through only an input unit.

Under this framework, for arbitrarily large networks, we are able to establish orbits within a high range of periodicity, or establish orbits that maximize the amount of time spent in a particular state. We develop a novel method of determining the number of code vectors to use in vector quantization of the state space of a deterministic dynamical system. This method looks directly at the influence the current vector quantization has on the dynamics of the system, as seen through the quantization. Since the system is deterministic, we can essentially see how accurately the system’s dynamics are being approximated. This makes our method more attractive for determining the number of code vectors than those typically used in clustering (e.g., [9]–[13]), as valuable information of the interactions between clusters is not used in these techniques.

We also show, using this method, that under the minimum input gain necessary for attaining temporal control of a network, the network’s dynamical complexity is reduced to approximately the same level, regardless of the size (i.e., dynamical richness) of the network, a phenomenon we refer to as “dynamical reduction”.

Since, by design, our dynamical system can become arbitrarily rich by increasing the size of the reservoir, it is possible that our results hold for a large number of families of non-linear dynamical systems, as the dynamics of the echo state network are linear combinations of non-linear functions. Of particular interest are biological control systems, as dynamical reduction in chaos control may have negative consequences on such systems where dynamical complexity is important, such as the brain.

The remainder of the paper is organized as follows: Section II provides background on echo state networks. Section III summarizes our control policy. Section IV summarizes the GNG state space approximation method. Section V elucidates the connection between input scaling and chaotic activity in a network. Section VI summarizes our method for quantifying the dynamical complexity of a system and provides evidence for dynamical reduction. Section VII presents results with simulations. Section VIII concludes the paper and suggests future work and applications.

II. THE ECHO STATE NETWORK

An echo state network (ESN) is a recurrently connected neural network often used in supervised learning problems. Extensive work has been done detailing the applications of such networks (e.g., [3], [5], [14]). Here, we briefly describe the basic ESN. An ESN consists of an input layer $\mathbf{u}$ of $M$ units connected to an $N \times N$ network of sparsely connected units, which can be connected to an output layer consisting of $P$ units. An $N \times M$ randomly weighted adjacency matrix between the input layer and the network is denoted by $\mathbf{W}^{in}$, and the $N \times N$ randomly weighted adjacency matrix of units within the network is denoted by $\mathbf{W}$, with $w_{ij}$ being the scalar that unit $j$’s output is multiplied by before being added to unit $i$’s internal state. Note that the graph is directed, so it
is not necessarily true that \( w_{ij} = w_{ji} \). The internal adjacency matrix is sparse and scaled by a parameter, \( \xi \). We sample \( w_{ij} \) from a mixture \( f_1 = \mathbb{U}[-\xi, \xi], f_2 = 1 \) at 0, with weights \( \rho \) and \( 1 - \rho \), respectively, where \( \rho \) is the probability that two units have a non-zero connection weight. We set \( \rho = 0.1, \xi = 1 \). Thus, for any unit \( i \), if we pick another unit \( j \), there is a probability of .1 that unit \( i \) inputs to unit \( j \), and if it does, the weight is sampled from \( \mathbb{U}[-1, 1] \).

Internal states of units, \( x \), are transformed by an activation function, which can be linear or non-linear, before inputting to other units. We use \( \tanh \) as a saturating non-linear activation function. The network units can be connected to an output layer \( y \) of \( P \) units by a \( P \times N \) matrix \( \mathbf{W}^{\text{out}} \). Output units then backpropagate into the network by an \( N \times P \) backpropagation matrix, \( \mathbf{W}^{\text{back}} \). Thus, the dynamics of the network can be expressed as follows:

\[
x(t + 1) = \tanh(\mathbf{W}^{\text{in}}\mathbf{u}(t + 1) + \mathbf{W}\mathbf{x}(t) + \mathbf{W}^{\text{back}}\mathbf{y}(t))
\]

(1)

Output layers are often used for supervised learning in echo state networks; in our study, we are interested in only the intrinsic dynamics of ESNs, so there is no output layer. Therefore, Equation 1 can be simplified to

\[
x(t + 1) = \tanh(\mathbf{W}^{\text{in}}\mathbf{u}(t + 1) + \mathbf{W}\mathbf{x}(t))
\]

(2)

In our study, one input unit is used. We choose the set of possible inputs to the network as a collection of \( k \) points equally spaced on and including the endpoints of \([-\Psi, \Psi]\), where \( \Psi \) is the input scaling parameter. Our choice of \( \Psi \) is discussed in Section V.

III. Q-LEARNING

We use the reinforcement learning algorithm Q-Learning [15] to achieve control goals. In reinforcement learning, an agent interacts with its environment, and at each time point, receives feedback from the environment in the form of a reward. Reinforcement learning aims to learn which actions to perform to maximize the total reward the agent will receive in the future. One can view the reward function as a mapping \( f : (S \times A) \rightarrow \mathbb{R} \) of state-action space onto the real numbers. At a given time \( t \), the agent exists in a state, \( s_t \). By performing an action, \( a_t \), the agent transitions to state \( s_{t+1} \) and receives a reward \( R_{t+1} \). Q-Learning uses value iteration to update the quality of a state-action pair at a time \( t \), \( Q(s_t, a_t) \):

\[
Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)],
\]

(3)

where \( \alpha \) is the learning rate, and \( \gamma \) is the discount factor, \( 0 \leq \gamma \leq 1 \), which modulates how future states are factored into \( Q \). For example, \( \gamma = 0 \) factors only immediate rewards into \( Q \), and \( \gamma = 1 \) considers all future rewards equally. We set \( \gamma = 0.9 \) and \( \alpha = 0.5 \).

When the goal is to achieve an arbitrary orbit of period \( T \), our reward function is as follows:

\[
R_{t+1}^T = -\vee \{ \mathbb{V} \\in \{ t+1, \ldots, t+T \} \ (s_t = s_{t'}) \vee (s_t \neq s_{t-T}) \},
\]

(4)

where \( \vee \) is logical OR. More simply put, Equation 4 evaluates to 0 if the value of \( s_t \) has not been repeated in the last \( T - 1 \) time steps, and if it has the same value as \( s_{t-T} \), which must be true for the system to be in a \( T \)-cycle. Otherwise, the reward is \(-1\).

If the goal is to maximize the time spent in a particular goal state, \( s^{\text{goal}} \):

\[
R_{t+1} = -\mathbb{1}(s_t \neq s^{\text{goal}}),
\]

(5)

where \( \mathbb{1}(\cdot) \) is the indicator function.

The agent follows a policy \( \pi : S \rightarrow A \) that determines which action the agent makes given a particular state. The value of a particular state at time \( t \), \( s_t \), under a policy \( \pi \) is defined to be

\[
V^\pi(s_t) = \mathbb{E} \left[ \sum_{k=1}^{\infty} R_{t+k} \gamma^{k-1} \right],
\]

(6)

where \( R_{t+k} \) is the expected reward at time \( t + k \) from following \( \pi \). Through value iteration, \( Q^\pi \) converges to \( V^\pi \) for \( 0 \leq \gamma < 1 \). We can define the optimal action under \( \pi \), given a particular state \( s \):

\[
a^*_\pi(s) = \arg\max_a Q^\pi(s, a).
\]

(7)

Often, the goal of Q-Learning is ultimately to approximate the optimal policy,

\[
\pi_* = \arg\max_\pi Q^\pi(S, A).
\]

(8)

In control systems, however, the ability to achieve control is more important than finding the absolute optimal policy. Thus, there must be a compromise between exploration and exploitation. To do so, we follow an \( \epsilon \)-greedy policy, in which a random action is chosen with probability \( \epsilon_t \), which starts at a set value and decays exponentially with time, encouraging exploration at the beginning of learning, and settling on a fixed policy as time progresses.

\[
P(\pi_{\epsilon_t}(s) = a) = \begin{cases} 1 - \frac{\epsilon_t}{\epsilon_T}, & a = a^* \\ \frac{\epsilon_t}{\epsilon_T}, & a \neq a^* \end{cases}
\]

(9)

IV. STATE SPACE APPROXIMATION

We use Q-Learning with tables, requiring the state space to be discretized. We initially take each unit’s activity as a collection of \( k \) points equally spaced on and including the endpoints of \([-\Psi, \Psi]\), with weights \( \Psi \) is the input scaling parameter. Our choice of \( \Psi \) is discussed in Section V.

\[
f(x) = \arg\min_{i \in \{1, 2, \ldots, k\}} d(x, w_i),
\]

(10)
where \( \{ \mathbf{w}_i \}_{i=1,2,...,k} \) is a set of vectors chosen to represent the state space, called code vectors, or a code book. We discuss the choice of \( k \), which we call \( N_{cv} \) in Section V. We consider \( d(\mathbf{a}, \mathbf{b}) = ||\mathbf{a} - \mathbf{b}||_2 \), assuming no prior knowledge of the state space manifold.

We use the growing neural gas algorithm (GNG) [2], a competitive Hebbian self-organizing map, for vector quantization of the state space. Briefly, in GNG, values of a probability distribution are independently sampled (in our case, network activity vectors are randomly sampled from the series of network activities generated during an exploration period (uniform random policy) before Q-Learning). The nearest (Euclidean) neighbor (NN) code vector and its topological neighbors (code vectors that share an edge with the NN code vector) move towards the sampled vector. Edges are added between the two NN code vectors during each sample. Edges decay in time to keep the topology of the code vectors up to date. If a code vector has no topological neighbors (edges), it is deleted. GNG starts with two randomly positioned code vectors and adds code vectors at a fixed rate. In our study, we set a limit to the total number of code vectors added, \( N_{cv} \).

We use the same parameters as Fritzke [2], but add a stopping criterion based on a threshold on the rate of change in the mean error (position difference) between a code vector and its NN datapoints. We run a uniformly random policy for some exploration time (\( O(N_{cv}|A|) \)) to gather a distribution of activity on which to run GNG.

V. INFLUENCE OF INPUT SCALING ON CHAOS

A minimal amount of input “power” (scaling) is needed to control the system. An input scaling below this minimal value will not sufficiently overpower the intrinsic dynamics of the system: the system will remain chaotic, and control is not possible. Chaotic activity can be avoided by scaling the internal connectivity matrix by \( \frac{1}{\sigma} \), where \( \sigma \) is the spectral radius (largest eigenvalue) of the internal connectivity matrix [3]; however, real-world systems do not conveniently scale in this fashion: it is usually not the case that the size of the network has a direct influence on the connectivity strength between units. Thus, we keep the internal weight matrix scaling, \( \xi \), constant across network sizes.

As the network dimension increases, a higher amount of “saturation” (how close each unit’s internal state is to the limits (±1) of the activation function \( \tanh \)) for each unit is required for the system to remain controllable, as the input must “overpower” a higher number of variable inputs from units in the reservoir. We define a unit to be saturated if its value (internal state passed through the \( \tanh \) saturating non-linearity) has an absolute value greater than 90% of its possible absolute value range. E.g., an internal state of \(-.95\) has an absolute value that is greater than 95% of all other possible absolute values (and is therefore saturated). A higher amount of saturation corresponds to a lower first derivative of the \( \tanh \) curve at that point, and thus, the dynamics of a unit are more stable at more saturated points. For each time step, we find the fraction of units that are saturated.

Figure 1 illustrates the need for larger networks to be more saturated by showing the fraction of saturated units during a standard temporal control simulation (achieving orbits of period 1-5). We attain qualitatively similar results for different percentile criteria for saturation. Interestingly, as the network dimension increases, saturation levels appear to divide between two distinct levels (Figure 1, black trace).

An arbitrarily large input scaling will overly diminish the dynamic complexity of the system, reducing it to a collection of binary units and rendering control trivial. Determining a sufficiently large input scaling that does not over saturate the system is difficult to determine and can depend on specific instantiations of networks. In addition, some scaling values will be sufficient for some temporal goals, but not others, with sufficiency not necessarily dependent on the period of the goal in question (e.g., attaining a 5 cycle does not imply that a 4 cycle will be attained under the same conditions).

Scaling values used for simulations in this paper were found by a bisection search algorithm that found the minimal input scaling (to the nearest integer) that achieved all specified period goals (usually periods 1-5). However, the precision of the minimal input scaling can be specified arbitrarily.

For the purposes of our present study, this method is sufficient. However, if we want to control an unknown system in real time, a different update rule must be adopted. A simple feedback rule (e.g., in supervised learning, using an input bias instead of an input scaling and finding the bias that minimizes the MSE between the network’s readout and the desired signal [16]) could also be used to appropriately scale the input: one might simply stimulate at the maximum input value and increase the scaling until a fixed point appears. We have found that a fixed point is usually accompanied by orbits of different periodicity if the input pattern is modified; however, this is not guaranteed. More work must be done to determine how scaling choice influences the periodicity of attainable orbits. In addition, if stimulation is penalized, and thus stimulation exploration is costly—as is the case in many biological applications—such an update method is not desirable.

VI. STATE SPACE APPROXIMATION AND DYNAMICAL COMPLEXITY

The number of code vectors used to approximate the state space and where they are located on the state space manifold are important for a policy to be efficient. Too few vectors under-represent the state space, and too many vectors lead to redundancy, inflating convergence time, exploration time, and memory storage requirements, as well as giving an inaccurate representation of the state space.

At first glance, finding the number of code vectors to use appears analogous to finding the number of clusters in a normal dataset, for which several techniques exist (e.g., [9]–[13]). However, it would be beneficial to leverage the deterministic nature of our system and come up with a technique based on a statistic that is directly related to the
The Gini index of a vector $\mathbf{z}$ is defined as:

$$G(\mathbf{z}) = 1 - 2 \sum_{k=1}^{N} \frac{z_k}{||\mathbf{z}||} \left( \frac{N - k + \frac{1}{2}}{N} \right)$$

For example, an infinitely long vector that contains only one non-zero component would have a Gini index of exactly 1, and any vector whose components are all the same value has a Gini index of exactly 0.

A dataset of network activities that is labeled with vector quantized state indices contains $|A|$ conditional transition probability vectors for each state, giving a total of $|A|N_N$ vectors and corresponding sparseness measures. For a single value to be associated with each quantity of code vectors used in GNG, we take the expectation of the Gini index over all conditional transition probability vectors with respect to states visited and actions performed. This prevents the sparsity measured to be biased by Gini indices of states that are not visited very often and whose conditional transition probability vectors therefore will be biased towards a high Gini index. To prevent a similar upward biasing of the Gini index as the dimensionality of $P(A, (S \times S))$ increases when we increase $N_N$, the number of time points we used in the calculation is $O(|A|N^2_N)$. The overall sparseness measure is therefore defined as

$$S(N_N) = \langle G(\hat{P}(a, (i, \cdot)))_{i,a} \rangle,$$  \hspace{1cm} (12)$$

where $\hat{P}$ is the empirical probability distribution and $\langle \cdot \rangle$ is the expectation over all $z$. We do this calculation on the same network activity data used to position GNG code vectors. We run GNG and then assign code vector state indices on the same data according to the GNG code vectors, and then finally run the transition probability sparseness calculation on the now-labeled data.

Sparseness should increase monotonically with the addition of more code vectors, as the system will always be more accurately described by more unique points. However, after enough code vectors have been added so that each region of activity has a code vector to distinguish it from other regions of activity, adding more code vectors leads to redundancy in representing states (i.e., new code vectors will be close to code vectors already present). So, if network activity is relatively clustered in state space, we can expect the sparseness to increase rapidly when initially adding code vectors, but then increase slowly after a certain number of code vectors (equal to the number of clusters) have been added. Thus, we should consider the point of highest curvature on a sparseness vs. number of code vectors graph to correspond to the number of code vectors necessary to describe the system. This heuristic is analogous to the “elbow method” [9] often used in k-means clustering [19]. That state space can be approximated as clusters is an assumption of our methodology.

Figure 2 shows simulation results of this calculation. Under the same input without additional scaling ($u$ restricted to $[-1, 1]$), as expected, conditional transition probabilities are less sparse under networks of higher dimension (Figure 2A). It is important to note that the “elbows” in Figure 2A appear at relatively low values of sparseness and then slowly increase with the addition of new code vectors. This is further demonstration of the dynamical richness of ESNs: such “diminishing returns” suggest the presence of manifold-like structures, even when the input space is discrete. We do not see such behavior under the controllable conditions (Figure 2B).
A surprising result is that under an input scaling necessary for control, the transition probability uncertainties are essentially the same for each network dimension (Figure 2B). Thus, during control, the dynamic complexity of each system is reduced to a common level—a phenomenon we call “dynamical reduction”. Since the point of highest curvature in each sparseness plot was at 50 GNG vectors, we use this as the number of code vectors for all of our control simulations.

VII. SIMULATION RESULTS

All simulations in this section were done on a 100 unit network (with an input scaling of 4), although we obtain similar results for networks of lower and higher dimensionality, consistent with our observation of dynamical reduction. We first wanted to know whether or not control of ESNs using only one input and no synaptic modification was possible. We find that, under sufficient input scaling, temporal (achieving a consistent periodic orbit of states) and spatial (maximizing the time spent in a particular state) control are both possible.

During each simulation, we run a random policy for an amount of time $O(N_{cv} |A|)$ (data not shown, as the states visited are random). We then run GNG on the network activity resulting from the random policy, which gives us the positions of the code vectors used for the state space approximation in our control algorithm. During control, after each time step, we convert the network state to our code vector state by finding the nearest (Euclidean) neighbor code vector and assigning that code vector’s index to the current state. Finally, we begin control with goals spanning time segments approximately 10 times longer than our exploration time. When the goal changes between each segment, all Q values are reset to 0, and $\epsilon$ is reset to $\epsilon_0$, which is 1 in our experiments.

Temporal control results—in which the goal is simply to achieve a state orbit of a specified period (reward function given by Equation (4))—under different frameworks are shown in Figure 3, where the GNG states (code vector state indices) during control are plotted in time. The goal periods are 1, 2, 3, 5, and 8 (in that order). Figure 3A demonstrates temporal control using our framework. All periodic goals are eventually achieved. Figure 3B shows the same network’s performance under a random vector quantization, where code vectors are given random positions within $[-1, 1]^N$. The system is able to achieve all period goals but the largest—a period of 8.

To show that control is not simply achieved through an arbitrary periodic input sequence (although, because of the echo state property [3], a temporal control input must be periodic), we show in Figure 3C that the resulting system under a random periodic input that is matched to the periodicity goal can achieve some low-period goals (e.g., fixed point and 2-cycle), but is unable to achieve stable periods for goals of higher periodicity (e.g., a 5- and 8-cycle). Here, code vectors are still determined by running GNG normally.

We can also change our goal to be a particular state and try to maximize the time the system is in that state (reward function given by Equation (5)). Figure 4 demonstrates the system establishing orbits around the goal state, thus maximizing the time spent in that state. Q-Learning essentially finds the orbit of lowest periodicity that visits the goal state. Figure 4A shows performance under our framework, which eventually finds an orbit that contains the goal state. We are unable to achieve performance discernible from a random policy using random code vectors (Figure 4B).

VIII. CONCLUSION

We have shown that, under sufficient input scaling, a single input to an arbitrarily large echo state network is sufficient to control the state of the network. Control extends to temporal goals, where the periodicity of the network’s activity is of interest, as well as spatial goals, where the goal is to maximize the time the network spends in a particular state. We find, however, that under sufficient input scaling to control a network, the dynamical complexity of the network is reduced to a common base level. This result would not have been apparent if we had not developed a method for
finding the optimal number of code vectors to use in GNG that depended directly on the dynamics of the resultant vector quantized state space. This method is especially important in the systems we deal with in this paper, where the original state space can have an arbitrarily large dimensionality, and the number of code vectors to use cannot be approximated a priori.

The idea of exploiting deterministic dynamics could be further pursued by modifying the actual GNG algorithm, e.g., inserting code vectors to areas with low transition sparseness, instead of areas with high intra-cluster error (as in normal GNG [2]), or combining both insertion techniques. This would lead to a more flexible dynamical representation of state space, where more complicated areas are represented by more vectors. A very similar approach has been used in less complicated systems, inserting code vectors preferentially near regions of high reward value [8]. However, the resultant representation would not provide a “balanced” representation of state space (as complex areas would be over-represented), which may be undesirable, given the specific goal (e.g., an orbit may actually stay relatively localized in state space). Hierarchical labeling may not work to alleviate this problem, as it would not be clear whether a particular area of space with a low sparseness is such because it is an area of high dynamical complexity, or rather, the area actually contains states that are far apart from each other in the state space manifold, and just close in Euclidean space. In addition, our method for calculating \( N_{cv} \) does not scale well with network dimension, so for large (\( \geq 250 \) units), accurate sparseness measures require a considerable amount of computational time. Modifying GNG to perform such a calculation many times would require even more computational time.

Since the dynamics of echo state networks are just linear combinations of non-linear functions, our results may extend to other real-world, non-linear systems. The most obvious application of control of an artificial neural network is control of a real neural system. Previous work [20] has shown that Q-Learning is capable of preventing and rescuing local epileptiform activity in brain slices using LFP signals and a single electrode. One application of our work could involve control of \textit{in vitro} or \textit{in vivo} systems with electrode current injections or laser pulses in optogenetic tissue. However, such biological systems contain many latent factors affecting system dynamics, and all neural activity might not be observable. Accordingly, we plan to extend our framework to restricted unit observability conditions.

On the other hand, in terms of practical applications, the subsequent dynamical reduction of each system raises questions about the effect of adaptive neurostimulation, in general. Reducing the dynamical complexity of a brain area could have detrimental effects to long- and short-term brain
Fig. 4: A. Simulation for 6 randomly chosen state proximity goals (black) using our framework. The actual (vector-quantized) states are shown in gray. For each state goal, we are able to achieve an orbit containing the goal state.

B. Simulation for the same goals in A, but using randomly chosen code vectors. Here, network activity is not discernible from that resulting from a random policy.

By using tables in Q-Learning, we ignore the relationship between different actions and states. However, one can think of the vector of unit activities at each time point as a point lying on the state space manifold, and inputs as curves on the manifold linking points. In this way, inputs of the same sign but different magnitude are represented as curves in the same direction on the manifold, but only different lengths. With a sufficient amount of exploration, interpolation techniques could be used to approximate the geometry of the state space manifold, and continuous action and/or state space reinforcement learning techniques could be used. Under our current framework, the dynamics we are able to achieve are bounded by the values and cardinality of the action space (the set of possible input values); by changing to a continuous action space framework, we would be able to harness much more of the dynamical richness offered by the system.

REFERENCES