Supervised Learning Algorithms for Controlling Underactuated Dynamical Systems

Bharat Monga\textsuperscript{a,*}, Jeff Moehlis\textsuperscript{a,b}

\textsuperscript{a}Department of Mechanical Engineering, Engineering II Building, University of California Santa Barbara, Santa Barbara, CA 93106, United States
\textsuperscript{b}Program in Dynamical Neuroscience, University of California Santa Barbara, Santa Barbara, CA 93106, United States

Abstract

Control of underactuated dynamical systems has been studied for decades in robotics, and is now emerging in other fields like neuroscience. Most of the advances have been in model based control theory, which has limitations when the system under study is very complex and it is not possible to construct a model. This calls for data driven control methods like machine learning, which has spread to many fields in the recent years including control theory. However, the success of such algorithms has been dependent on availability of large datasets. Moreover, due to their black box nature, it is challenging to analyze how they work, which may be crucial in applications where failure is very costly. In this paper, we develop two related novel supervised learning algorithms. The algorithms are powerful enough to control a wide variety of complex underactuated dynamical systems, and yet have a simple and intelligent structure that allows them to work with a sparse data set even in the presence of noise. Our algorithms output a bang-bang (binary) control input by taking in feedback of the state of the dynamical system. The algorithms learn this control input by maximizing a reward function in both short and long time horizons. We demonstrate the versatility of our algorithms by applying them to a diverse range of applications including: switching between bistable states, changing phase of an oscillator, desynchronizing a population of synchronized coupled oscillators, and stabilizing an unstable fixed point\textsuperscript{1}. For most of these applications we are able to reason why our algorithms work by using traditional dynamical systems and control theory. We also compare our learning algorithms with some traditional control algorithms, and reason why our algorithms work better.

Keywords: Supervised Learning, Underactuated Dynamical Systems, Bang bang control, Coupled Oscillators, Machine Learning, Binary Classification

1. Introduction

Underactuated dynamical systems are systems with fewer actuators/controls than the dimensionality of the state space of the system. Such systems are ubiquitous in a variety of fields including physics, chemistry, biology and engineering. There have been numerous advances made on controlling such systems, with much of the work in robotics \cite{8, 38, 84}. Control in other applications, especially biology \cite{31, 43, 10, 10}, is on the rise as it provides promising treatment strategies for several disorders such as Parkinson’s disease, cardiac arrhythmias, and jet lag. Most of these control methods, both in robotics and biology, are based on the traditional model based control theory and optimal control.

Such methods work well when it is possible to model the dynamics of the system accurately, which is very difficult as the systems become complicated, especially in neuroscience applications where the dynamics

\textsuperscript{1}Implementation of the algorithms for some of these examples is available at \url{https://github.com/bharatmonga/Supervised-learning-algorithms}.

\textsuperscript{*}Corresponding author

\textit{Email addresses: monga@ucsb.edu} (Bharat Monga), \textit{moehlis@engineering.ucsb.edu} (Jeff Moehlis)
of a single neuron may change rapidly depending on the response from other neurons in the network. Even if an accurate model could be built to describe the dynamics of such a system, developing a classical model based control for such an underactuated system is a challenging task. If the parameters of the system change with time, or if the model doesn’t describe the dynamics accurately, the theoretical control guarantees like stability and boundedness may not apply in real systems [13, 11]. This calls for the development of data driven control algorithms that can learn to control the system without explicitly using a model.

Artificial intelligence algorithms are able to learn to control dynamical systems by using deep neural networks. Such algorithms have been used for a long time [14, 2], but with the availability of large data sets, improvement in deep learning architectures, optimization methods, and cheap computation costs, their use is on the rise [16]. However, the success of such algorithms has been largely dependent on availability of large datasets [17], which can be limited in fields like neuroscience where the cost of obtaining human/animal brain data is very high. Moreover, the black box nature of such methods makes their analysis difficult. Such analysis would be important in tasks where failure is very costly. Another limitation of such methods is their inability to take advantage of the inherent dynamics of the system to achieve the task, which limits their performance.

All these limitations call for a new machine learning control algorithm that doesn’t rely on large amounts of data, is easy to understand, and can take advantage of the underlying dynamics in achieving the task. In this article, we have developed two related novel supervised learning algorithms based on these three goals. Our algorithms are powerful enough to control a wide variety of complex underactuated dynamical systems, and yet have a simple structure so one can understand how they work using dynamical systems and control theory foundations. Their simple yet intelligent structure also allows them to effectively achieve the control objective by training on a sparse data set, even in the presence of noise. Our algorithms output a bang-bang (binary) control input by taking in feedback of the state of the dynamical system. The algorithms learn this control input by maximizing a reward function in both short and long time horizons. We demonstrate the versatility of our algorithms by applying them to a diverse range of underactuated dynamical systems including: switching between bistable states, changing the phase of an oscillator, desynchronizing a population of synchronized coupled oscillators, and stabilizing an unstable fixed point of a dynamical system. For most of these applications we are able to reason why our algorithms work by using traditional dynamical systems and control theory. We compare our algorithms with traditional control algorithms and reason why our algorithms work better, especially because they autonomously learn to take advantage of the underlying system dynamics in achieving the control objective. We carry out a robustness analysis to demonstrate the effectiveness of our algorithms even in the presence of noise.

This article is organized as follows. In Section 2, we develop our supervised learning algorithms and the binary classifier to output a binary control. We demonstrate our first supervised learning algorithm by controlling underactuated bistable dynamical systems in Section 3 and compare our algorithm to a fully actuated control. In Section 4 we illustrate the effectiveness of our second supervised learning algorithm by controlling the phase of a single oscillator and comparing the algorithm to a model based optimal control algorithm. We further demonstrate the versatility of our second supervised learning algorithm by using it to desynchronize a population of synchronized coupled oscillators in Section 5. In Section 6 we apply our second algorithm to stabilize an unstable fixed point of an underactuated dynamical system, and compare the algorithm to a model based control algorithm. To demonstrate the applicability of our algorithms in a real setting, we show how their intelligent structure allows them to perform well in the presence of noise. Section 8 summarizes our work and suggests future extensions and tools. Appendix A lists the mathematical models used in this article. In Appendix B we give background on phase reduction relevant for the model based control comparison and validation of our second learning algorithm in Sections 4 and 5 respectively.

2. Supervised Learning Algorithm

We consider an underactuated dynamical system with an additive control input $\pi(x(t))$ as

$$\frac{d}{dt} x(t) = F(x(t)) + [\pi(x(t)), 0_{n-1}]^T, \quad x(t) \in \mathbb{R}^n, \quad (1)$$
where \(0_{n-1}\) is an \(n-1\) dimensional zero vector. Thus, the control input depends on the full state of the dynamical system, and only directly affects the first state of the dynamical system. The control input is binary in nature having two values \(\{u_1 > 0, u_2\}\), which can be chosen differently for different applications. For our first algorithm, we take \(u_2 = 0\), and thus the control can be thought of having an “ON” state with the value \(u_1\) and an “OFF” state with value 0. For our second algorithm, we take the control to be a bang bang control with \(u_2 = -u_1\). Both algorithms learn the control input \(\pi(x(t))\) as a function of the state to achieve a particular control objective. They do so by learning from the data generated by sampling a model describing the underlying dynamics of the system. To demonstrate our algorithm in this article, we use an analytical model \((F(x(t)))\) which generates our training data. In case a model is not available in an application, one can still use the same algorithms by obtaining training data by direct measurement of the states of the system at different times. Below we describe our supervised learning algorithms in more detail.

2.1. Supervised Learning Algorithm 1

Our first supervised learning algorithm outputs a “ON” and “OFF” binary control input. The algorithm learns what control input to output to achieve a certain control objective by maximizing a reward function \(R(x(t))\) which needs to be carefully designed to achieve a control objective in a particular application.

We sample a state \(x(0)\) randomly from the state space of the dynamical model of the system, and evolve the state forward in time for short time \(\Delta t\) with control state OFF. If \(R(x(\Delta t)) \geq R(x(0))\), we set the control policy for state \(x(0)\) as OFF. If it is not, we again evolve the initial state forward for the same time but with control ON, and compare \(R(x(\Delta t))\) for both control ON and OFF. Whichever control policy maximizes the reward is set for the sampled state \(x(0)\). We repeat the process \(N\) times by sampling more initial states randomly. The algorithm is summarized below:

**Algorithm 1**

```
Initialize \(X\) as zeros(N,length(\(x\))) and \(U\) as zeros(N,1)
for i=1,N do
    Randomly sample \(x(0)\)
    Compute \(x(\Delta t)\) and \(R(x(\Delta t))\) with control OFF
    if \(R(x(\Delta t)) \geq R(x(0))\) then
        Set policy for \(x(0)\) as OFF
    else
        Compute \(x(\Delta t)\) and \(R(x(\Delta t))\) with control ON
        if \(R(x(\Delta t))\) with control ON > \(R(x(\Delta t))\) with control OFF then
            Set policy for \(x(0)\) as ON
        else
            Set policy for \(x(0)\) as OFF
        end if
    end if
end for
Assign \(X[i,\cdot]\) as \(x(0)\) and \(U[i,\cdot]\) as the policy
return \(X, U\)
```

Such an algorithm takes advantage of the underlying dynamics by letting the trajectories evolve without any control and only switching “ON” the control when necessary. This makes our algorithm highly energy efficient. Such an algorithm is very suitable for controlling bistable dynamical systems where the objective is for the trajectory to converge to a particular stable state of the system, or to switch from one stable state to another. The control can switch OFF when the trajectory enters the region of attraction of the desired stable state, and let the dynamics take the trajectory to the desired state.
2.2. Supervised Learning Algorithm 2

Our second supervised learning algorithm outputs a bang-bang control input which can be used to control a variety of dynamical systems, including coupled oscillators. The algorithm learns what control input to output to achieve a certain control objective by maximizing a reward function $R(x(\Delta t))$, which needs to be carefully designed to achieve a control objective in a particular application.

We sample a state $x(0)$ randomly from the state space of the dynamical model of the system, and evolve the state forward in time for short time $\Delta t$ with both control $u_1$ and $-u_1$. In both scenarios we let the state evolve further in time with zero control until some event occurs and measure the timing of this event. The reward $R(x(\Delta t))$ is dependent on the timing of this event. Whichever policy ($u_1$ or $-u_1$) maximizes this reward is set for that sampled state. We repeat the process $N$ times by sampling more states randomly. The algorithm is summarized below:

Algorithm 2

| Initialize $X$ as zeros(N,length($x$)) and $U$ as zeros(N,1)
| for i=1,N do
|     Randomly sample $x(0)$
|     Compute $x(\Delta t)$ and $R(x(\Delta t))$ with control $u_1$
|     Compute $x(\Delta t)$ and $R(x(\Delta t))$ with control -$u_1$
|     if $R(x(\Delta t))$ with control $u_1 \geq R(x(\Delta t))$ with control -$u_1$ then
|         Set policy for $x(0)$ as $u_1$
|      else
|         Set policy for $x(0)$ as -$u_1$
|      end if
|     Assign $X[i, :]$ as $x(0)$ and $U[i, :]$ as the policy
| end for
| return $X, U$

Such control is useful when the objective is to converge to an unstable state of the system, because the control needs to stay “ON” (be non-zero) the whole time even when the control objective has been realized, since the trajectory will go back to the stable state otherwise. Here as well the underlying dynamics of the system play a role in our learning algorithm: to determine the control input for a particular initial state, we let the dynamics evolve the trajectory until an event occurs.

Both these algorithms generate data comprising a set of $N$ sampled states of the dynamical system $X$, and a set of the corresponding control inputs $U$. However, we need to know the control input $\pi(x(t))$ as a function of a general trajectory $x(t)$ of the system, since the trajectory is not restricted to these sampled states. This is achieved with our binary classifier, that takes an input as the state of the dynamical system $x(t)$ and outputs the corresponding control policy $\pi(x(t))$ based on this generated data.

2.3. Binary Classifier

Our supervised learning algorithms generate data comprising a set $X$ of $N$ sampled states of the dynamical system, and a set of the corresponding control inputs $U$. Based on this information we build a locally weighted binary classifier that takes as input as the instantaneous state of the dynamical system $x(t)$, and outputs the corresponding control policy $\pi(x(t))$ to be applied at that instant.

We assign each element of the set $X$ with a weight

$$w_i(x(t)) = \exp\left(-\frac{|x(t) - X_i|^2}{2\tau}\right), \quad i = 1, 2, \ldots, N,$$

such that $w_i(x(t))$ is a Gaussian distribution centered at $X_i$ with standard deviation $\tau$. The binary classifier then outputs the control policy $\pi(x(t))$ as a function of the weighted states $X_i$.
where $X_i$ represents the $i^{th}$ sampled state stored in the set $X$. Thus a sampled state is given a higher weight if its closer to $x(t)$, and a lower weight if it is further away from $x(t)$. These weights are normalized so that

$$\sum_{i=1}^{N} w_i(x(t)) = 1.$$  

For the first algorithm, the classifier outputs $\pi(x(t)) = u_1$ (“ON”) if

$$\sum_{i=1}^{N} w_i(x(t))U_i > 0.5u_1, \tag{2}$$

and $\pi(x(t)) = 0$ (“OFF”) otherwise. Similarly for the second algorithm, the classifier outputs $\pi(x(t)) = u_1$ if

$$\sum_{i=1}^{N} w_i(x(t))U_i > 0, \tag{3}$$

and $\pi(x(t)) = -u_1$ otherwise.

With $\pi(x(t))$ defined to be the output of this binary classifier, we simulate the dynamical system from \ref{eq:1} starting from a random initial condition and find that our supervised learning algorithms are able to achieve the desired control objectives, while simultaneously maximizing the designed reward functions. The entire algorithm is depicted in the flowchart in Figure 1.

![Flowchart of the Learning Algorithm](image)

Figure 1: Flowchart of the Learning Algorithm.

Note that in our algorithm, the sets $X$, $U$ need to be computed only once for a given dynamical system, whereas the control input $\pi(x(t))$ is computed by the binary classifier at every timestep.

3. Bistable Dynamical Systems

Bistability is widely found in neural systems \cite{25} and cardiac arrhythmia \cite{7}, and is used in digital electronics for storing binary information, in mechanical switches for transitioning between ON and OFF states, and in multivibrators, Schmitt trigger circuits, and even optical systems \cite{5}. It is the key mechanism for understanding several cellular processes including those associated with the onset and treatment of cancer \cite{24}. In this section we apply our first supervised learning algorithm to control underactuated bistable dynamical systems. The control objective is for the trajectory to converge to a particular stable fixed point.
of the system starting anywhere in the state space (including in the basin of attraction of the other stable state). Such a control objective is relevant for several applications such as biocomputing, gene therapy, and treatment of cancer \cite{13,9}, among others.

### 3.1. Duffing Oscillator

With the Duffing oscillator \cite{10,18}, we consider the class of bistable dynamical systems having two stable fixed points (\(x^s_1, x^s_2\)), and an unstable fixed point (\(x^u\)). The control objective is for the trajectory to converge to \(x^{s2}\) starting anywhere in the state space. The Duffing oscillator is given as:

\[
\begin{align*}
\dot{x} &= y + \pi(x(t)), \\
\dot{y} &= x - x^3 - \delta y.
\end{align*}
\]

For \(\delta > 0\), the system has two stable fixed points \(x^{s1} = (-1, 0)\) and \(x^{s2} = (1, 0)\), and an unstable fixed point \(x^u = (0, 0)\), all shown in Figure 2. We take \(\delta = 0.1\) in our simulations.

#### 3.1.1. Learning Algorithm

We choose our reward function to be the negative of the Euclidean distance between the current state and the desired state:

\[
R(x(t)) = -||x(t) - x^{s2}||.
\]

Thus the control will make the trajectory converge towards the desired fixed point while increasing the reward to 0. To converge to \(x^{s2}\) starting anywhere in the state space, we use our learning algorithm to generate a control policy. The ON (resp., OFF) state of the control policy corresponds to a value of \(u_1 = 4\) (resp., 0). We randomly sample \(N = 50\) points for generating the sets \(X\), \(U\), and choose \(\Delta t = 0.001\) and \(\tau = 0.4\).

#### 3.1.2. Results

The generated control policy is shown in the left panel of Figure 2. Blue open circles (resp., black \(\times\)’s) represent elements of the set \(X\) where the control policy given by elements of the set \(U\) is OFF (resp., ON). The green (resp., red) region is where output \(\pi(x(t))\) of the binary classifier is OFF (resp., ON). A controlled and an uncontrolled trajectory starting from same \(x(0)\) is shown in the right panel of Figure 2. As can be seen in this figure, the control algorithm gradually converges the trajectory to \(x^{s2}\) by turning the control ON a few times, whereas the uncontrolled trajectory converges to \(x^{s1}\).

The algorithm generates an energy efficient control policy as the policy is OFF 60.41% of the total time it takes to drive the trajectory within a ball of radius of 0.45 in the region of attraction of \(x^{s2}\). We investigate the robustness of our learning algorithm by testing it on 1000 randomly generated initial conditions, and in all the 1000 cases, the control algorithm is able to converge the trajectories to \(x^{s2}\), achieving 100% accuracy. Choosing \(N\) is the crucial task in our learning algorithm. We start with a small \(N\) and keep it increasing until the algorithm achieves 100% effectiveness. \(N = 50\) points turns out to be appropriate for the Duffing oscillator as choosing a lower number of points leads to underfitting, and choosing a higher number of points leads to overfitting. Both underfitting and overfitting can have a detrimental effect on the accuracy of the learning algorithm.

### 3.2. Reduced Hodgkin-Huxley model

With the reduced Hodgkin-Huxley model \cite{27,20,12}, we consider the class of bistable dynamical systems having a stable periodic orbit \(x^{s1}(t)\), an unstable periodic orbit \(x^{u}(t)\), and a stable fixed point \(x^{s2}\). The model is given as

\[
\begin{align*}
\dot{v} &= (I - g_{Na}(m_\infty(v))^3(0.8 - n)(v - v_{Na}) - g_K n^4(v - v_K) - g_L (v - v_L)) / c + \pi(x(t)), \\
\dot{n} &= a_n(v)(1 - n) - b_n(v)n,
\end{align*}
\]

where \(v\) is the trans-membrane voltage, and \(n\) is the gating variable. \(I\) is the baseline current, which we take as 6.69 \(\mu A/cm^2\), and \(\pi(x(t))\) represents the applied control policy. For the rest of the parameters, see Appendix A.1. In the absence of control input, the system is bistable having \(x^{s1}(t)\) with period 14.91 ms, \(x^{s2}(t)\) with period 14.33 ms, and \(x^u = (-61.04, 0.38)\), all shown in Figure 3.
3.2.1. Learning Algorithm

The control objective is to make the trajectory converge to the stable fixed point starting anywhere in the state space. Here as well we choose the reward function (4). Without any control input, a trajectory starting outside $x^u(t)$ will converge to $x^s_1(t)$, and a trajectory starting inside $x^u(t)$ will converge to $x^s_2$. To converge to the stable fixed point starting anywhere in the state space, we use our learning algorithm to generate a control policy. The ON (resp., OFF) state of the control policy corresponds to a value of $u_c = 15$ (resp., 0). We sample $N = 1000$ points for generating the sets $X, U$, and choose $\Delta t = 0.001$, and $\tau = 0.001$. Because the two state variables $v, n$ scale differently, it is important to normalize them for calculating the reward function. This is also important for the binary classifier to work effectively, since it is based on the Euclidean norm. To do this, we subtract from each element of the set $X$ the mean of the set and then divide each element by the variance of the set. We subtract the same mean and divide by the same variance from the state $x(t)$ that goes in calculating the reward function and also the binary classifier.

3.2.2. Results

The generated control policy is shown in the left panel of Figure 3. As shown in this figure, the learning algorithm indicates that it is better to have control ON in the left part of the state space in order to maximize the reward function. In all other regions, the learning algorithm indicates that the control policy should be OFF. Since the control policy is ON in only a small region of the state space, we need to sample 1000 points to accurately determine this region. A controlled trajectory using this policy is shown in the right panel of Figure 3. The learning algorithm is able to converge the trajectory to the stable fixed point $x^s_2$ by bypassing the unstable periodic orbit $x^u(t)$. The algorithm generates an energy efficient control policy as the policy is OFF 23.81% of the time it takes for the algorithm to drive the trajectory inside $x^u(t)$. We investigate the robustness of our learning algorithm by testing it on 1000 randomly generated initial conditions, and in all the 1000 cases, the algorithm is able to converge the trajectories to $x^s_2$, achieving 100% effectiveness. Note that the learning algorithm has no information about the periodic orbits and fixed points of the system, it only works to maximize the reward function.
3.2.3. **Comparison with fully actuated control**

To further demonstrate energy efficiency of our learning algorithm, we compare it with a fully actuated feedback control given as

$$\frac{d}{dt}x(t) = F(x(t)) + U(x(t)), \quad x(t) \in \mathbb{R}^n,$$

$$(5)$$

$$U(x(t)) = -F(x(t)) - 0.2(x(t) - x^2),$$

$$(6)$$

which also converges the trajectory to the stable fixed point in the same time frame as our learning algorithm. However, the energy required by this algorithm calculated as $\int_0^T ||U(x(t))||_2^2 dt$ comes out to be more than 3 orders of magnitude larger compared to the energy taken by the control obtained from our learning algorithm. This is because our learning algorithm takes advantage of the natural dynamics of the system to drive the trajectory close to the desired point, and turns the control ON only for a short amount of time when its really needed. In contrast, the feedback based control is ON the whole time, even when the trajectory reaches inside the unstable periodic orbit.

4. **Phase Control of an Oscillator**

In this section, we use our second algorithm to control a class of underactuated dynamical systems having a stable limit cycle solution $x^*(t)$. We seek to maximally increase or decrease the phase of the limit cycle solution by using a bang-bang type control input. The motivation behind such a control objective comes from controlling neurons, where one might want a neuron to spike as quickly as possible subject to a constraint on the magnitude of the allowed input current; this constraint can be due to hardware limitations and/or concern that large inputs might cause tissue damage. Thus instead of thinking in terms of maximally increasing the phase, one can instead think in terms of maximally decreasing the neuron’s spike time.

4.1. **Model**

To demonstrate our algorithm, we consider the 3-dimensional thalamic neuron model [37] for the oscillatory behavior of neurons in the thalamus:
\begin{align*}
\dot{v} &= -\frac{I_L(v) - I_{Na}(v,h) - I_K(v,h) - I_T(v,r) + I_b}{C_m} + \pi(x(t)), \\
\dot{h} &= \frac{h_\infty(v) - h}{\tau_h(v)}, \\
\dot{r} &= \frac{r_\infty(v) - r}{\tau_r(v)},
\end{align*}

where the state $x(t)$ is the tuple $(v,h,r)$, $v$ is the transmembrane voltage, and $h, r$ are the gating variables of the neuron. $\pi(x(t))$ represents the applied current as the control input. For details of the rest of the parameters, see Appendix A.2. With no control input, these parameters give a stable limit cycle $x^s(t)$ with period $T = 8.40$ ms shown in red in Figure 4.

4.2. Learning Algorithm

Here the control objective is to maximally decrease the spike time of the neuron, meaning we want the oscillation to end sooner than it naturally would. We set the reward function as the negative of the neuron’s next spike time (the time when the transmembrane voltage $v(t)$ reaches a maximum):

$$R(x(t)) = -t_{\text{spike}}.$$  

We sample 100 states randomly along the limit cycle and evolve them with both positive and negative control inputs for time $\Delta t = 0.001$, and then evolve them further with zero control input until the neuron spikes. Whichever control input attains the minimal $t_{\text{spike}}$ (maximizes the reward function) is selected as control policy for that sampled state. We choose $\tau = 0.01$. Because the state variables $v, h, r$ have different dynamic ranges, we normalize the set $X$ and the state at every time step similar to in Section 3.2.1.
4.3. Results

The generated control policy, along with the controlled trajectory, is shown in Figure 4. As seen in this figure, most of the sampled states need to have a positive control in order to maximize the reward function. This is evident from the left panel of Figure 5 which plots the corresponding control input. Because of the control, the neuron spikes ($v$ reaches its maximum) in $t_{\text{spike}} = 7.49 ms$ which is 10.82% decrease in its natural spike time of 8.40 ms. Thus our algorithm is able to achieve the control objective while keeping the controlled trajectory close to the stable limit cycle solution (see Figure 4).

4.4. Model based control comparison

The dynamics of neural oscillations are highly nonlinear and high dimensional, which makes a model based control formulation very challenging. Phase reduction, a model reduction technique valid close to the limit cycle (see Appendix B for more details), can overcome these challenges. The neuron spike time control problem was solved as an optimal control problem in [33, 31] using phase reduction, which also resulted in a bang bang control with control input given as

$$\pi(x(t)) = -\text{sign}[Z(\theta)]u_1 \text{ for decreasing } t_{\text{spike}},$$

(11)

where $u_1$ is the bound chosen by the user and $Z(\theta)$ is the phase response curve (see Appendix B for more details) which is a periodic function of $\theta$. Such a control works well, except when the bound $u_1$ is large, where the controlled trajectory can diverge far away from the limit cycle, decreasing the accuracy of phase reduction and making the control based on phase reduction ineffective. Effectiveness of such a control also relies heavily on accurate measurement of phase response curve, which may not be possible.

We find that our learning based control outputs a control input very similar to the above model based control, both shown in the left panel of Figure 5 for $u_1 = 1$. We compute $t_{\text{spike}}$ as a function of the bound $u_1$ and find that our learning based algorithm does slightly better than the model based algorithm in decreasing $t_{\text{spike}}$ (shown in the right panel of Figure 5). Both controls are able to decrease $t_{\text{spike}}$ more as $u_1$ increases.

5. Desynchronization of a Population of Coupled Oscillators

Populations of coupled oscillators are ubiquitous in applications from physics, chemistry, biology, and engineering [34, 23, 44, 20]. The collective behavior of such oscillators varies, and includes synchronization,
desynchronization, and clustering in various scenarios. Pathological synchronization of neural oscillations in the thalamus and the subthalamus region of the STN brain region is hypothesized to be one of the causes of motor symptoms for essential and parkinsonian tremor, respectively [19, 21]. Deep brain stimulation (DBS), an FDA approved treatment, has proven to alleviate these symptoms [3, 4] by stimulating the thalamus or the STN brain regions with a high frequency, (relatively) high energy pulsatile waveform, which has been hypothesized to desynchronize the synchronized neurons; see, e.g., [41, 43]. This has motivated researchers to come up with efficient model dependent control techniques [39, 32, 42, 28, 30] to desynchronize the neural oscillations, but also consume less energy, thus prolonging the battery life of the stimulator and preventing tissue damage or side effects caused by the pulsatile stimuli.

5.1. Model

Inspired by such treatment of parkinsonian and essential tremor, we employ our algorithm to desynchronize an initially synchronized population of $M$ coupled thalamic neural oscillations. We consider the 3-dimensional thalamic neuron model [37] for each individual oscillator with added all-to-all electrotonic coupling:

$$\dot{v}_i = -I_L(v_i) - I_{Na}(v_i, h_i) - I_K(v_i, h_i) - I_T(v_i, r_i) + I_b + \frac{1}{C_m} \sum_{j=1}^{M} \alpha_{ij}(v_j - v_i),$$

$$\dot{h}_i = \frac{h_{\infty}(v_i) - h_i}{\tau_h(v_i)},$$

$$\dot{r}_i = \frac{r_{\infty}(v_i) - r_i}{\tau_r(v_i)},$$

where $x(t)$ represents the full state ($3 \times M$ dimensional) of the oscillator population. Here, $i = 1, \ldots, M$, where $M$ is the total number of oscillators in the neuron population. $v_i$ is the transmembrane voltage, and $h_i, r_i$ are the gating variables of the $i$th neural oscillator. $\alpha_{ij}$ is the coupling strength between oscillators $i$ and $j$, which are assumed to be electrotonically coupled [15] with $\alpha_{ij} = 0.01$ for $j \neq i$ and $\alpha_{ii} = 0$ for all $i$. $\pi(x(t))$ represents the applied current as the control policy. For details of the rest of the parameters, see [Appendix A.2]. Note that the same control input $\pi(x(t))$ is applied to all of the oscillators. With no control input, these parameters give a synchronized oscillator population with period $T = 8.40$ ms.

5.2. Learning Algorithm

We index the individual neural oscillators in the order in which they spike, thus neuron 1 spikes first and neuron $M$ spikes last. We set the reward function as the absolute value of spike time difference of neuron 1 and $M$:

$$R(x(t)) = |t_{\text{spike}_1} - t_{\text{spike}_M}|.$$  

(15)

Since the oscillator population is initially synchronized, this reward is initially a small positive number as all neurons spike very close to each other. We aim to desynchronize the population by maximally increasing this reward function. We consider $M = 51$ oscillators in the synchronized population and sample 51 states along the synchronized oscillation. Since the state of the oscillator population is very large ($3 \times M$), we take the mean across the population to reduce the dimension of our set $X$. The $i$th element of the set $X$ is given as

$$X_i = \frac{\sum_{j=1}^{M} (v_j, h_j, r_j)}{M}.$$  

(16)

We evolve the oscillator population with both positive and negative control inputs for time $\Delta t = 0.001$, and evolve them further with zero control input until all neurons in the population spike. Whichever policy attains the maximum reward function is selected for that sampled state $X_i$. 

11
The binary classifier takes as input the full high dimensional state of the oscillator population. It then computes the mean of the state across the population and compares it with the sampled mean states to output a control policy \( \pi(x(t)) \). Because the mean of the states \( v_i, h_i, r_i \) scale differently, it is important to normalize them for the binary classifier to work effectively, since it is based on the Euclidean norm. Thus we normalize the set \( \mathcal{X} \) and the mean state at every time step similar to in Section 3.2.1. We choose \( \tau = 0.01 \).

5.3. Results

The generated control policy shown in Figure 6 gives a positive control input in the bottom left region of oscillation and a negative control in the top right region of the oscillation. The same figure also plots a model based control policy discussed below. Figure 7 plots the results of desynchronization of a thalamic neuron population by our learning algorithm. As shown in both the left and right panels of the figure, the control policy from our learning algorithm is able to desynchronize an initially synchronized thalamic neuron population in about 90 ms while keeping the oscillators close to the initially synchronized oscillation. It may seem that the population is largely synchronized from the right panel of Figure 7 but that is not the case. Since the oscillators spend most of their time near the top of the limit cycle, one naturally observes more of them near the top of the limit cycle even though they are evenly spread out in time (and not space). This becomes clear from the left panel of Figure 7.

5.4. Model based validation of control policy

Here we analyze why the policy predicted by our learning algorithm works. Consider an oscillator population comprised of just 2 oscillators whose dynamics evolve according to phase reduction as

\[
\begin{align*}
\dot{\theta}_1 &= \omega + Z(\theta_1)u(t), \\
\dot{\theta}_2 &= \omega + Z(\theta_2)u(t)
\end{align*}
\]

The dynamics of their phase difference \( \phi = \theta_1 - \theta_2 \) can be written as (cf. [12])

\[
\dot{\phi} = Z'(\theta)u(t)\phi + \mathcal{O}(\phi^3),
\]

Figure 6: Thalamic synchronized population oscillation: The closed curve is the synchronized oscillation. Small blue circles (resp., black ×’s) represents elements of the set \( \mathcal{X} \) where the control policy given by elements of the set \( \mathcal{U} \) is \(-u_1\) (resp., \(u_1\)). The oscillation is plotted in red (resp., green ) where \( Z'(\theta) \) is negative (resp., positive).
where $\theta = 0.5(\theta_1 + \theta_2)$ is the mean of the two oscillators’ phases, and $Z'(\theta)$ is the derivative of the phase response curve with respect to $\theta$. If the oscillators are synchronized then their phase difference $\phi \approx 0$, thus higher order term in equation (19) can be ignored and the equation can be rewritten as

$$\dot{\phi} = Z'(\theta)u(t)\phi. \quad (20)$$

To desynchronize these two synchronized oscillators the coefficient of $\phi$ in the above equation should be positive. This can be achieved if $u(t)$ is of same sign as $Z'(\theta)$. This is exactly what our policy predicts, as is shown in Figure 6. The policy predicts the control to be positive in the region of oscillation where $Z'(\theta)$ is positive, and it predicts the control to be negative in the region of oscillation where $Z'(\theta)$ is negative, thus explaining why our algorithm is able to desynchronize the oscillator population.

6. Stabilizing an Unstable Fixed Point

In this section we apply our second learning algorithm to stabilize an unstable fixed point of an underactuated dynamical system. This control objective is one of the oldest studied control theory problems that is employed in several fields including robotics, electrochemical systems, and treatment of cardiac arrhythmias. To demonstrate this, we consider the Lorenz system given as:

$$\begin{align*}
\dot{x} &= \sigma(y - x) + \pi(x(t)), \\
\dot{y} &= rx - y - xz, \\
\dot{z} &= xy - bz.
\end{align*} \quad (21)$$

In the absence of control input with parameters $\sigma = 10$, $b = 8/3$, $r = 1.5$, the system is bistable with $x^1 = (-1.15, -1.15, 0.5)$, $x^u = (0, 0, 0)$, and $x^{s2} = (1.15, 1.15, 0.5)$, all shown in Figure 5.

6.1. Learning Algorithm

The control objective is for a trajectory to converge to the unstable fixed point $x^u$ starting anywhere in the state space. We choose our reward function to be the negative of the Euclidean distance between current state and the desired unstable fixed point

$$R(x(t)) = -||x(t) - x^u||. \quad (24)$$
Thus the control will make the trajectory converge towards the desired fixed point while increasing the reward to 0. To converge to $x^u$ starting anywhere in the state space, we use our learning algorithm to generate a control policy. We take $u_1 = -u_2 = 5$, and sample $N = 1000$ points for generating the sets $X$, $U$. We choose $\Delta t = 0.001$ and take the binary classifier parameter $\tau = 5$.

6.2. Results

The generated control policy is shown in the left panel of Figure 8. Blue open circles (resp., black $\times$’s) represent elements of the set $X$ where the control policy given by elements of the set $U$ is -5 (resp., 5). A controlled trajectory using our learning algorithm and an uncontrolled trajectory starting from same $x(0)$ are shown in the right panel of Figure 8. The learning algorithm converges the trajectory to $x^u$, whereas the uncontrolled trajectory converges to $x^{s1}$. In doing so, the learning based control consumes 150 units of control energy ($\int_0^6 \pi(x(t))^2 \, dt$). We investigate the robustness of our learning algorithm by testing it on 1000 randomly generated initial conditions, and in all the 1000 cases, the learning based control algorithm is able to converge the trajectories within a ball of radius 0.09 units centered at $x^u$, achieving 100% effectiveness.

6.3. Comparison with Lyapunov based control

To demonstrate energy efficiency of our learning algorithm, we compare it with Lyapunov-based control to stabilize $x^u$. Consider the following positive definite Lyapunov function

$$\begin{align*}
V(t) &= \frac{1}{2} x(t)^2 + \frac{1}{2} y(t)^2 + \frac{1}{2} z(t)^2, \\
\dot{V}(t) &= -2\sigma x(t)^2 - 2y(t)^2 - 2bz(t)^2 + 2x(t) \left( u(t) + (\sigma + r)y(t) \right),
\end{align*}$$

Its time derivative is given as

$$\dot{V}(t) = -2\sigma x(t)^2 - 2y(t)^2 - 2bz(t)^2 + 2x(t) \left( u(t) + (\sigma + r)y(t) \right),$$

where $u(t)$ takes the place of $\pi(x(t))$ in equation (21). Then by taking $u(t) = -(\sigma + r)y(t)$, one gets a negative definite time derivative of the Lyapunov function. Thus by the Lyapunov theorem, this control
Figure 9: Flowchart of the Learning Algorithm with added noise.

asymptotically stabilizes the unstable fixed point $x^u$. The control trajectory based on this control is plotted in blue in the right panel of Figure 8. As seen in the figure, the Lyapunov-based control is able to converge the trajectory towards the unstable fixed point as well. But in doing so, it consumes 1176.8 units of control energy ($\int_0^6 u(t)^2 dt$), which is almost 8 times the energy consumed by our learning based control. This is partly because our learning based control uses the inherent system dynamics to control the trajectory, as the controlled trajectory seems to stay close to the uncontrolled trajectory. In contrast, the Lyapunov based control drives the trajectory far away before it converges to $x^u$, thus it ends up consuming much more energy.

7. Robustness to Noise

We have demonstrated the effectiveness of our algorithms in several scenarios in which the algorithms were based on data generated from a deterministic dynamical model. However, real data measured from an experimental setup will be noisy. In order for our algorithms to work in an experimental setup it is imperative to investigate their performance when the data is corrupted with noise. We do that by considering the Duffing oscillator in the bistable parameter regime from Section 3.1. The control objective is still for the trajectory to converge to $x^{s2}$ starting anywhere in the state space.

7.1. Learning Algorithm

To replicate the effect of noise in an experimental setup, we use exactly the same parameters as before to generate the sets $\mathcal{X}$, $\mathcal{U}$ and corrupt the set $\mathcal{X}$ by adding Gaussian white noise with mean 0 and standard deviation $\sigma$, resulting in the set $\tilde{\mathcal{X}}$. Thus each element in the dataset will be offset from its true value. We also add Gaussian white noise with the same mean and standard deviation to the state $x(t)$ resulting in $\tilde{x}(t)$, which the binary classifier takes as input at every time step. This accounts for the noise in estimation of the state by the classifier in a real system. The flowchart from Figure 1 with added noise is modified and shown in Figure 9.

7.2. Results

The generated control policy corrupted with noise of standard deviation $\sigma = 0.2$ is shown in the left panel of Figure 10. Blue open circles (resp., black ×’s) represent elements of the noise corrupted set $\tilde{\mathcal{X}}$ where
Figure 10: Duffing Oscillator with noise ($\delta = 0.1$): Solid (resp., open) black circles represent $x^{s_1}$, $x^{s_2}$, (resp., $x^u$). In the left panel, open blue circles (resp., black $\times$’s) represent elements of the set $\tilde{X}$ where the control policy given by elements of the set $\mathcal{U}$ is OFF (resp., ON). The green (resp., red) region is where the output $\pi(\tilde{x}(t))$ of the binary classifier is OFF (resp., ON). The elements of the original set $X$ are plotted in white. In the right panel, the trajectory starts in the region of attraction of $x^{s_1}$, and converges to $x^{s_2}$ (resp., $x^{s_1}$) with (resp., without) control. When the control policy is ON (resp., OFF), the trajectory is plotted in black (resp., blue). The uncontrolled trajectory is plotted in red.

the control policy given by elements of the set $\mathcal{U}$ is OFF (resp., ON). The green (resp., red) region is where output $\pi(\tilde{x}(t))$ of the binary classifier is OFF (resp., ON). The elements of the original set $X$ are plotted in white to show the shifting of the elements due to the noise. Besides shifting the elements, the addition of white noise blurs the decision boundary between the ON and OFF policy region. A controlled and an uncontrolled trajectory starting from the same $x(0)$ is shown in the right panel of Figure 10. As can be seen in this figure, the control algorithm converges the trajectory to $x^{s_2}$ even though it has been corrupted by adding noise both to the training dataset and to the input of the binary classifier. On the other hand, the uncontrolled trajectory converges to $x^{s_1}$. Note that the controlled trajectory follows a different route when compared to the noiseless case from figure 2. This is because the noise distorts the decision boundary between ON and OFF states of the policy, resulting in a slightly different path.

We investigate the robustness of our learning algorithm in the presence of noise by testing it on 1000 randomly generated initial conditions, and in all the 1000 cases the control algorithm is able to converge the trajectories to $x^{s_2}$, achieving 100% accuracy. Choosing $\sigma$ does have an effect on the effectiveness of our algorithm. We observed that for $\sigma$ up to 0.2 the algorithm achieves 100% effectiveness. This is because of the way our binary classifier is designed. It outputs a policy based on all the elements of the dataset $X$ instead of just a few nearest neighbors (see equation 2). Because of noise, the elements get shifted to $\tilde{X}$, but since all the elements are used in the summation in equation 2 for making a decision, the slight shifting of each element is averaged out by this summation. For $\sigma > 0.25$, the algorithm’s effectiveness starts dropping as the shifts in $\tilde{X}$ become too big to be offset by the summation of $N = 50$ points. Thus, if the noise intensity is very high, the algorithm can perform well by taking a higher number of points ($N$) in the dataset to effectively offset the shifting of the elements.

8. Conclusion

In this article we have developed two novel supervised learning algorithms to control a range of underactuated dynamical systems. The algorithms output a bang bang (binary) control input to achieve the desired control objectives which maximize a reward function. A simple yet intelligent structure allows the algorithms to be energy efficient as they autonomously learn to take advantage of the inherent dynamics.
We demonstrated the versatility of our algorithms by applying them to a diverse range of applications including: switching between bistable states, changing the phase of an oscillator, desynchronizing a population of synchronized coupled oscillators, and stabilizing an unstable fixed point. For most of these applications we were able to reason why our algorithms work by using traditional dynamical systems and control theory. We also compared our algorithms to some traditional nonlinear model control algorithms and showed that our algorithms work better. We also carried out a robustness analysis to demonstrate the effectiveness of our algorithms even with noisy data.

We simulated various dynamical models to generate data for training our supervised learning algorithms. In an experimental setting such an algorithm can be similarly implemented by stimulating the system with binary control inputs at different states of the system and determining which control input works best for the different sampled states, and ultimately using that information in constructing a binary classifier. The data generated from an experimental setting might be corrupted with noise, thus to demonstrate the potential of our algorithm in a real setting we showed that our algorithm works even in the presence of noise. The unique structure of our binary classifier comes to the rescue by negating the adverse effects of noise. Note that having an additive control input doesn’t restrict our algorithms. Since the algorithms work by maximizing the reward function, the structure of control input coming into the dynamics does not matter. In the future, we plan to explore how to modify our algorithm if some of the states are not observable, and how to adapt our algorithm for very high dimensional dynamical systems.

Appendix A. Models

In this appendix, we give details of the mathematical models used in this article.

Appendix A.1. Reduced Hodgkin-Huxley model

Here we give the reduced Hodgkin-Huxley model [27, 20, 12] used in Section 3.2:

\[
\begin{align*}
\dot{v} &= (I - g_{Na}(m_{\infty}(v))^3(0.8 - n)(v - v_{Na}) - g_{K}n^4(v - v_{K}) - g_{L}(v - v_{L})) / c + \pi(x(t)), \\
\dot{n} &= a_{n}(v)(1 - n) - b_{n}(v)n,
\end{align*}
\]

where \(v\) is the trans-membrane voltage, and \(n\) is the gating variable. \(I\) is the baseline current, which we take as 6.69 \(\mu A/cm^2\), and \(\pi(x(t))\) represents the applied control current.

\[
\begin{align*}
a_{n}(v) &= 0.01(v + 55)/(1 - \exp(-(v + 55)/10)), \\
b_{n}(v) &= 0.125 \exp(-(v + 65)/80), \\
a_{m}(v) &= 0.1(v + 40)/(1 - \exp(-(v + 40)/10)), \\
b_{m}(v) &= 4 \exp(-(v + 65)/18), \\
m_{\infty}(v) &= a_{m}(v)/(a_{m}(v) + b_{m}(v)), \\
c &= 1, \ g_{L} = 0.3, \ g_{Na} = 120, \ v_{Na} = 50 \\
g_{K} &= 36, \ v_{K} = -77, \ v_{L} = -54.4 \ I = 20.
\end{align*}
\]

Appendix A.2. Thalamic neuron model

The thalamic neuron model is given as

\[
\begin{align*}
\dot{v} &= \frac{-I_{L}(v) - I_{Na}(v, h) - I_{K}(v, h) - I_{T}(v, r) + I_{b}}{C_{m}} + \pi(x(t)), \\
\dot{h} &= \frac{h_{\infty}(v) - h}{\tau_{h}(v)}, \\
\dot{r} &= \frac{r_{\infty}(v) - r}{\tau_{r}(v)}.
\end{align*}
\]
Appendix B. Phase Reduction

Phase reduction is a classical technique to describe dynamics near a limit cycle. It works by reducing the dimensionality of a dynamical system to a single phase variable $\theta$ \cite{44,23}. Consider a general $n$-dimensional dynamical system given by

$$\frac{dx(t)}{dt} = F(x(t)), \quad x(t) \in \mathbb{R}^n, \quad (n \geq 2).$$

Suppose this system has a stable periodic orbit $x^*(t)$ with period $T$. For each point $x^*$ in the basin of attraction of the periodic orbit, there exists a corresponding phase $\theta(x^*)$ such that

$$\lim_{t \to \infty} \left| x(t) - x^* \left( t + \frac{T}{2\pi} \theta(x^*) \right) \right| = 0,$$

where $x(t)$ is the flow of the initial point $x^*$ under the given vector field. The function $\theta(x)$ is called the asymptotic phase of $x$, and takes values in $[0, 2\pi)$. For neuroscience applications, we typically take $\theta = 0$ to correspond to the neuron firing an action potential. Isochrons are level sets of this phase function, and it is typical to define isochrons so that the phase of a trajectory advances linearly in time both on and off the limit cycle, which implies that

$$\frac{d\theta}{dt} = 2\pi \equiv \omega$$

in the entire basin of attraction of the limit cycle. Now consider our underactuated system given by equation (1). Phase reduction can be used to reduce this system to a one-dimensional system given by \cite{45, 22, 6, 31}:

$$\dot{\theta} = \omega + \mathcal{Z}(\theta) \pi(x(t)).$$

Here $\mathcal{Z}(\theta)$ is the first component of the gradient of phase variable $\theta$ evaluated on the periodic orbit and is referred to as the (infinitesimal) phase response curve (PRC). It quantifies the effect of an infinitesimal control input on the phase of a limit cycle.

Acknowledgment

This work was supported by National Science Foundation Grant No. NSF-1635542.
