2021 American Control Conference (ACC)
New Orleans, USA, May 25-28, 2021

State Constrained Stochastic Optimal Control Using LSTMs

Bolun Dai[1], Prashanth Krishnamurthy[1], Andrew Papanicolaou[2], Farshad Khorrami[1]

Abstract— **In this paper, we propose a new methodology for state constrained stochastic optimal control (SOC) problems. The solution is based on past work in solving SOC problems using forward-backward stochastic differential equations (FBSDE). Our approach in solving the FBSDE utilizes a deep neural network (DNN), specifically Long Short-Term Memory (LSTM) networks. LSTMs are chosen to solve the FBSDE to address the curse of dimensionality, non-linearities, and long time horizons. In addition, the state constraints are incorporated using a hard penalty function, resulting in a controller that respects the constraint boundaries. Numerical instability that would be introduced by the penalty function is dealt with through an adaptive update scheme. The control design methodology is applicable to a large class of control problems. The performance and scalability of our proposed algorithm are demonstrated by numerical simulations.**

I. Introduction

Optimal control has wide applications in robotic control [1], navigation [2], to name a few. With the availability of higher computational power and powerful optimization software, such as SNOPT [3], optimization algorithms have been applied to increasingly complex control problems. Optimal control casts the control problem in terms of cost functions that are addressed through numerical optimization techniques. In reality, systems also contain many uncontrolled inputs, such as measurement noise and external forces with unknown distributions, which if not taken into consideration in design, will lead to performance degradation. To address these uncontrolled inputs, a set of optimal control problems, namely stochastic optimal control, have been considered, which takes disturbances or measurement noise into consideration more explicitly.

The solution to a SOC problem involves solving a nonlinear partial differential equation (PDE), known as the Hamiliton-Jacobi-Bellman (HJB) equation. One approach is to approximate the state dynamics by linear [4] or quadratic [5] systems. However, this requires fine-grained time discretization or specialized linearization techniques. When the system is of high dimension, solving the HJB equation becomes difficult due to the curse of dimensionality. An alternative to linearization is sampling-based methods, such as Markov-Chain Monte Carlo (MCMC) approximation [6] to the HJB equation and forward-backward stochastic differential equations (FBSDE) [7]. However, numerical methods for MCMC-based methods are difficult to scale due to their reliance on predefined grids for value function backward propagation [6]. They also suffer from compounding errors in least square approximations for FBSDE-based methods [7]. Work has also been done in state constrained SOC problems, [8] translated the problem into a state-constrained target problem and used its backward reachable sets to describe the value function. However, the efficacy of [8] was not demonstrated on a physical system.

Deep learning has been successfully applied to several application domains in recent years. One application of deep learning is to serve as an alternative to numerical methods in solving high-dimensional and nonlinear PDEs. In [9], DNNs are used to solve PDEs, such as Black-Scholes, HJB, and Allen-Cahn equations. However, in [9], only problems with available analytic solutions and uncontrolled state dynamics are tested. Following this line of work, DNNs have been utilized to solve FBSDEs for SOC problems. By using DNNs, we can learn the initial condition and then propagate forward the solution to the backward stochastic differential equation (BSDE). Additionally, solving the FBSDE using neural network (NN) based methods resolves the aforementioned compounding-error issue. The basic formulation of using LSTMs to solve FBSDEs was shown in [10], along with adding control constraints. However, the paper did not consider state constraints, which occur in most physical control problems. Another branch of deep learning, namely deep reinforcement learning (DRL), has also been deployed to solved optimal control problems. One significant difference between deep FBSDE based methods and DRL is the control in deep FBSDE based methods are given as functions of the value function whereas in DRL, the value function is either used as a baseline to reduce variance [11] or a state-action value function is used [12] to obtain control actions.

In this paper, we propose a SOC setting for nonlinear systems that incorporates state constraints. In Section II, the state constrained SOC formulation is given. In Section III, the deep FBSDE algorithm is presented along with modifications to include control saturation. In Section IV, an algorithmic solution is provided for adding state constraints. This results in a log-barrier-like method. In Section V, the NN architecture and overall algorithm are presented. In Section V, an adaptive update scheme to the state constraint penalty function is shown, which enhances stability during NN training. In Section VI, we show the efficacy of our approach on a cart-pole system under two different state constraints, namely cart movement and system energy. Finally, the paper is concluded and potential future research directions are presented.

This work is supported in part by the NSF under grant DMS-1907518.

[1]Control/Robotics Research Laboratory, Electrical & Computer Engineering Department, Tandon School of Engineering (Polytechnic Institute), New York University, Brooklyn, NY, 11201 bd1555@nyu.edu, prashanth.krishnamurthy@nyu.edu, khorrami@nyu.edu.

[2]Department of Mathematics, College of Sciences, North Carolina State University, Raleigh, NC, 27695 apapani@ncsu.edu.

II. Problem Formulation

In this section, we outline the optimal control problem under state constraints. A system with dynamics that involves stochastic processes can be described using a stochastic differential equation (SDE) as follows

$$dx(t) = f(x(t),t)dt + G(x(t),t)u(t)dt + \Sigma(x(t),t)dw(t) \tag{1}$$

with initial condition $x(0) = x_0$ and $w(t) \in \mathbb{R}^\nu$ being a standard Brownian motion. The states are denoted by $x \in \mathbb{R}^n$, time by $0 < t < T < \infty$, and the control input by $u \in \mathbb{R}^m$. In (1), $f : \mathbb{R}^n \times [0,T] \to \mathbb{R}^n$ represents the drift, $G : \mathbb{R}^n \times [0,T] \to \mathbb{R}^{n\times m}$ represents the control influence, and $\Sigma : \mathbb{R}^n \times [0,T] \to \mathbb{R}^{n\times\nu}$ represents the diffusion (influence of the Brownian motion on the state evolution). The state constrained SOC problem is to find a controller $u(t)$ that minimizes an objective function $J^u(x,t) \in \mathbb{R}^+$ under a set of state constraints. The objective function is defined as

$$J^u(x,t) = \mathbb{E}\Big[g(x(T)) + \int_t^T \Big(q(x(s)) + \frac{1}{2}u(s)^T R u(s)\Big)ds \Big| x(t) = x\Big] \tag{2}$$

where $g : \mathbb{R}^n \to \mathbb{R}^+$ is the terminal state cost, $q : \mathbb{R}^n \to \mathbb{R}^+$ is the instantaneous state cost, and the control cost matrix is $R \in \mathbb{R}^{m\times m}$, which is positive definite. We consider state constraints in the following form

$$c_{\min} \le c_s(x) \le c_{\max} \tag{3}$$

where $c_s(x) \in \mathbb{R}^r$ is a vector of functions of the state, and $c_{\min} \in \mathbb{R}^r$ and $c_{\max} \in \mathbb{R}^r$ represent the lower and upper bounds of $c_s(x)$ component-wise, respectively. An example of this kind of constraint is a box constraint on states, i.e., wherein $c_s(x) = x$. Additionally, control saturation (with $U_{\max} \in \mathbb{R}^m$) is introduced, which has the form of

$$u \in \mathcal{U} = \{u \mid |u_i| \le U_{i,\max}\} \tag{4}$$

where u_i and $U_{i,\max}$ are the i^{th} elements of u and $U_{\max}$, respectively. The value function $V(x,t) \in \mathbb{R}^+$ is defined as

$$V(x,t) := \inf J^u(x,t) \tag{5}$$

where the $\inf$ is computed over all control signals $u(.)$ over the time interval $(t,T]$ satisfying the constraint (4).

III. Deep FBSDE Formulation

To solve the state constrained SOC problem, a FBSDE formulation is used. In this section, we first consider the unconstrained SOC problem. Later in this section, we introduce the deep FBSDE formulation that is modified to handle control saturation. Modifications required to handle state constrained SOC problems will be introduced in Section IV. Starting from (2), we can use Dynkin's formula [13] along with Bellman's principle [14] to arrive at the HJB equation

$$V_t(x,t) + \mathcal{L}V(x,t) + h(x,V_x,t) \tag{6a}$$
$$V(x,T) = g(x) \tag{6b}$$

where V_t is the partial derivative of V with respect to t. The generator function $\mathcal{L}V(x,t)$ is defined as

$$\mathcal{L}V = \frac{1}{2}\text{trace}(\Sigma\Sigma^T V_{xx}) + f^T V_x \tag{7}$$

where V_x and V_{xx} denote the first and second order partial derivatives of V w.r.t. x, respectively. The Hamiltonian is

$$h(x,V_x,t) = \inf_{u\in\mathcal{U}} \left(q(x) + (G(x,t)u)^T V_x(x,t) + \frac{1}{2}u^T R u\right). \tag{8}$$

Using first-order conditions to solve for the optimal control in the Hamiltonian yields

$$u^*(x,t) = -R^{-1}G^T(x,t)V_x(x,t). \tag{9}$$

Define $G(x(t),t) = \Sigma(x(t),t)\Gamma(x(t),t)$, where $\Gamma : \mathbb{R}^n \times [0,T] \to \mathbb{R}^{\nu\times m}$. This form of G captures the characteristic that the range of G belongs to the range of Σ, which excludes the case of a channel containing control input without noise. This representation aligns with the fact that no noiseless control signal exists in reality.

Additional to this basic formulation, the control inputs are saturated as shown in (4). The saturation is introduced using the sig function as in [10], which is defined as

$$\text{sig}(v) = \frac{2}{1+e^{-v}} - 1. \tag{10}$$

The optimal control action is then calculated as

$$u^*(x,t) = U_{\max} * \text{sig}(-R^{-1}G^T(t,x)V_x) \tag{11}$$

with the control saturated between $[-U_{\max}, U_{\max}]$. In (11), "$*$" represents element-wise multiplication. For the control saturated system, a new value function is introduced

$$\mathbb{E}\left[g(x(T)) + \int_t^T \left(q(x(s)) + \sum_{i=1}^m S_i(u_i(s))\right) ds \Big| x(t) = x\right]. \tag{12}$$

The new control cost $S_i(u_i)$ is defined as

$$S_i(u_i) = c_i \int_0^{u_i} \text{sig}^{-1}\Big(\frac{v}{U_{i,\max}}\Big)dv \tag{13}$$

with constant weights c_i, dummy variable for integration v. For this control saturated system, the Hamiltonian will become

$$h(x,V_x,t,u^*) = q(x) + V_x^T G(x,t)u^*(x,t) + \sum_{i=1}^m S_i(u_i^*). \tag{14}$$

Following the formulation in [10], we can solve for (11) using a FBSDE, which has the form of

$$dy(t) = \Big(-h\big(x(t), \Sigma^T(x(t),t)V_x(x(t),t;\theta),t\big) + V_x^T\big(x(t),t;\theta\big)G\big(x(t),t\big)u\big(x(t),t\big)\Big)dt + V_x^T\big(x(t),t;\theta\big)\Sigma\big(x(t),t\big)dw(t) \tag{15a}$$

$$dx(t) = \Big(f\big(x(t),t\big) + G\big(x(t),t\big)u\big(x(t),t\big)\Big)dt + \Sigma\big(x(t),t\big)dw(t) \tag{15b}$$

$$u(t) = U_{\max} * \text{sig}(-R^{-1}G^T\big(x(t),t\big)V_x(x(t),t;\theta) \tag{15c}$$

$$y(0) = V(\phi) \tag{15d}$$
$$dy(0) = V_x(\phi) \tag{15e}$$
$$x(0) = x_0. \tag{15f}$$

The deep FBSDE method [10] introduces a NN with parameters θ to estimate $V_x(x(t_n), t_n)$ at each time step, which is denoted as $V_x(x(t_n), t_n; \theta)$. Using the $V_x(x(t_n), t_n)$ estimation and following (11), we can obtain the control action u_n^*. The initial value function estimation $V(x(t_0), t_0)$ and its partial derivative $V_x(x(t_0), t_0)$ are also learned via trainable weights ϕ; with some abuse of notation, we denote these learned models as $V(\phi)$ and $V_x(\phi)$. This enables the forward propagation of the backward part in the FBSDE. The original continuous finite time horizon problem is discretized into N time steps ($T = N\Delta t$). This gives us $\Delta y(t_n)$ and $\Delta x(t_n)$. Using a numerical integration scheme as in [9], we can find the value function estimation at the last time step $V(x(t_N), t_N)$ along with the terminal state $x(T)$. The loss function of the DNN is chosen such that $\|V(x(t_N), t_N) - g(x(T))\|_2$ is minimized. Note that $V(x(t_N), t_N)$ is equivalent to $y(T)$.

IV. State Constraint Formulation

In this section, we present our formulation to incorporate state constraints on top of the control saturated FBSDE formulation. At each time instant, a NN is used to estimate V_x. Then, V_x is used to calculate the optimal control that minimizes the cost-to-go. Additionally, the optimal controller of the state constrained SOC problem needs to satisfy the state constraints. We can modify the cost function, such that the cost-to-go is minimized only when the state constraints are respected. This can be achieved by adding a "soft" constraint in the form of a penalty, which increases the cost when the state is outside the constraint boundary. This approach is preferred over using a sigmoid-like function to force the state dynamics into a predefined range. Using the sig function, the dynamics would be altered using a "fake" saturation which is not present in the real dynamics — hence, training on such an altered dynamic model would not yield a controller that functions properly under the real dynamics. When adding the penalty, states inside the constraint boundary should not be penalized, but for states outside the constraint boundary, a large penalty should be given. Therefore, the penalty function should be close to zero inside the constraint boundary and a large number outside. The function

$$p(x) = \frac{L}{1 + e^{-k(c_s(x) - c_{\max})}} - \frac{L}{1 + e^{-k(c_s(x) - c_{\min})}} + L - \frac{2L}{1 + e^{-k(\mu - c_{\max})}} \tag{16}$$

satisfies the aforementioned requirements. $L \in \mathbb{R}^+$ is a scalar determining the maximum value of the penalty, $k \in \mathbb{R}^+$ determines the steepness of the boundary (larger k leads to steeper boundaries), as shown in Figure 1, and $\mu = 0.5(c_{\max} + c_{\min}) \in \mathbb{R}^r$ represents the mid-point of the constraint region. The proposed penalty function consists of two parts: the first part is the two logistics functions, which gives the "U"-like shape; the second part moves the minimum value of the penalty function to zero.

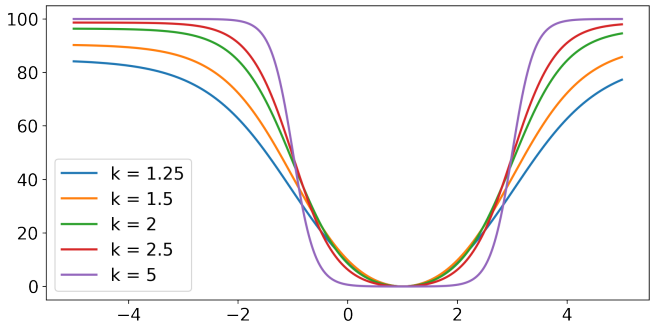

Fig. 1. $p(x)$ in (16) under different values of k, with $\mu = 1$, $L = 100$, $c_s(x) = x$, $c_{\min} = -1$, $c_{\max} = 3$ and x being a scalar. It can be seen that as k increases, the boundaries gets increasingly steep, and the values of $p(x)$ inside the constrained region $(-1, 3)$ go to zero.

Algorithm 1 Soft state constraint update

Given:
k: Boundary steepness, δ: Change of boundary steepness, β: Update threshold, γ: Threshold change ratio, l: Iteration number, Δ: Boundary steepness change acceleration, η: Update interval, η': Max interval, Δ_δ: Threshold change acceleration;
if state trajectory not inside constraint boundary **then**
  if $l \bmod \eta = 0$ **then**
    $\bar{c} = \frac{1}{N}\sum_{l=1}^{\eta} c(x_l)$;
    $\sigma_c^2 = \frac{1}{N}\sum_{l=1}^{\eta} (c_l - \bar{c})^2$;
    if $\sigma_c < \beta$ or $l \bmod \eta' = 0$ **then**
      $k, \delta, \beta, \gamma = k + \delta, \delta - \Delta_\delta, \gamma\beta, \gamma + \Delta$
  if $\delta < 0$ **then**
    $\delta = 0$;
  if $\gamma > 1$ **then**
    $\gamma = 1$;
else
  End Algorithm

Using the penalty function $p(x)$, the new state cost at each time instant becomes

$$c(x) = q(x) + p(x). \tag{17}$$

Note, when choosing parameters for $p(x)$, ideally we would pick larger k and L values to ensure a steeper boundary. After applying a penalty function, the Hamiltonian becomes

$$h(x, V_x, t, u^*) = c(x) + V_x^T G(x,t) u^*(x,t) + \sum_{i=1}^{m} S_i(u_i^*) \tag{18}$$

where u^* is defined in (11). Thus, state constraints can be applied by simply swapping the Hamiltonian defined in (14) to the form in (18).

V. State Constrained Deep FBSDE Controller

In this section, we present the state constrained deep FBSDE algorithm, an overview of the corresponding NN

architecture, and an adaptive update scheme to the state constraints that greatly enhances training stability.

A. Algorithmic Design

The time horizon T is discretized into N time steps. At each time step n, we use a NN parameterized by θ. The input to the NN at each time step is the current state x_n^m (and the hidden state from the previous time step H_{n-1}^m when using LSTMs), where m represents the m-th sample in the mini-batch. The output of the NN at each time step is $V_x^m(x(t_n), t_n))$, which is then used to calculate the control action using (9). Following (15), we can obtain the next value function estimation and the next state using Euler's integration. After the last time step, the loss is computed as

$$L = \frac{1}{M}\sum_{m=1}^{M} \|g(x_N^m) - y_N^m\|_2^2 + \lambda\|\theta\|_2^2 \tag{19}$$

where λ determines the weight of the regularization term. The initial value function estimate $V(x(t_0), t_0)$ and its partial derivative $V_x(x(t_0), t_0)$ are parameterized by ϕ_V (if using LSTMs, initial hidden states H_0 are parameterized by ϕ_H). The initial state ξ is fixed. We use a batch size of M during training. The NN is trained for a total L iterations using the Adam optimizer under a varying learning rate. All of the weights are initialized using the Xavier normal initializer.

B. Neural Network Architecture

Both dense-layer based and LSTM based NN architectures have been used in [10]. The benefits of using a LSTM based architecture are two-fold. Firstly, the weights are shared among time steps, compared to a dense-layer based architecture resulting in reduced number of weights. Secondly, for dense-layer based methods, weights at each time step are step specific, making it difficult to scale to long horizons. For LSTM based architectures, we can train for a shorter time horizon than during testing, given the state distribution for longer time horizons are the same. The NN architecture illustrated in Figure 2 is inspired by [10]; thus, only a brief introduction is provided. A LSTM-based architecture is used to tackle the vanishing gradient problem in long time horizons. Additional to the basic LSTM model, we have a forward part which represents the flow of state dynamics and a backward part representing the flow of the value function.

C. Adaptive Update Scheme

In the initial stage of training, parts of many trajectories lie outside of the constraint boundary. If a steep boundary is in use, it will result in a very large gradient and cause numerical instabilities in training even if the gradient is clipped. If the steepness of the boundary is fixed at a small value, the penalty of violating the constraint is not significant enough for the learning algorithm to achieve the required state constraint. Therefore, an adaptive update scheme is needed to gradually increase the boundary steepness k in order to avoid exploding gradients while shaping the controller behavior.

For a given k, we train the DNN until the performance of the controller cannot be further improved, then we update k. To determine the performance of the controller, we can use the square root of state cost variance $\sigma_c \in \mathbb{R}$ over a given amount of iterations. The state cost is defined in (17); its square root variance decreases as the learning algorithm converges to a solution. Define the threshold of determining convergence as $\beta \in \mathbb{R}^+$. Once σ_c becomes smaller than β, we update k. We check for the condition $\sigma_c < \beta$ every η iterations; if the condition is not satisfied after η' iterations, k will also be updated (may be stuck in a local minimum).

During the training process, the variance decreases. Thus, β should also be decreased after each update. To make the decrease in β smoother, we let the "acceleration," $\Delta \in \mathbb{R}^+$, become negative. Similarly, the acceleration of the increase of k is made negative, which is denoted by $\Delta_\delta \in \mathbb{R}^+$, leading to more fine-grained changes in k at later stages of training. Algorithm 1 shows the update scheme for k and the overall algorithm is given in Algorithm 2.

VI. Experiments

In this section, we show the efficacy of our control algorithm on the cart-pole for a swing-up task under two different state constraints: cart movement and system energy. For both systems, the trained model is evaluated over 256 trials. All experiments are carried out using TensorFlow.

A. Cart-pole Swing-Up Task I

The task for the cart-pole system is to swing the pole up from downward position and stabilize it at the upright position. The cart-pole dynamics are given as

$$(M+m)\ddot{x} + mL\sin\theta\dot{\theta}^2 - mL\cos\theta\ddot{\theta} = u \tag{20}$$

$$mL^2\ddot{\theta} - mL\cos\theta\ddot{x} - mgL\sin\theta = 0 \tag{21}$$

where x is the cart position, θ is the pendulum angle with respect to the downward position, $M = 1.0$kg is the cart mass, $m = 0.01$kg is the pole mass (point mass at the tip), and the pole length L is 0.5m. The initial state is at origin and the target state is $[0, \pi, 0, 0]^T$, with the states being $[x\ \theta\ \dot{x}\ \dot{\theta}]^T$. The control input u is saturated at ± 10N. The cart position x and velocity $\dot{x}$ are constrained at ± 1.5m and ± 2.5m/s, respectively. The time horizon is chosen to be 2.5 sec and the time step is $\Delta t = 1/110$ sec. Disturbances are applied to the linear and angular velocities with a discount factor of 0.25. We use the cost function

$$\sum_{i=1}^{N}\Big[\frac{1}{2}\mathbf{X}_i^T Q\mathbf{X}_i + S_i(u_i) + p(\mathbf{X}_i)\Big] \tag{22}$$

where $\mathbf{X}_i$ is the difference between the current state and the target state. Q is the cost weight matrix for the state, and S_i is defined in (13). $p(\cdot)$ is the penalty function that incorporates the state constraint. The state dynamics under the control generated by the trained model are shown in Figure 3. For the constrained stochastic optimal controller, the state lies within the constraint boundary for all time. Figure 4 shows that without the presence of a state constraint, the cart position and velocity do violate constraints significantly. After applying the state constraint, the effect can be

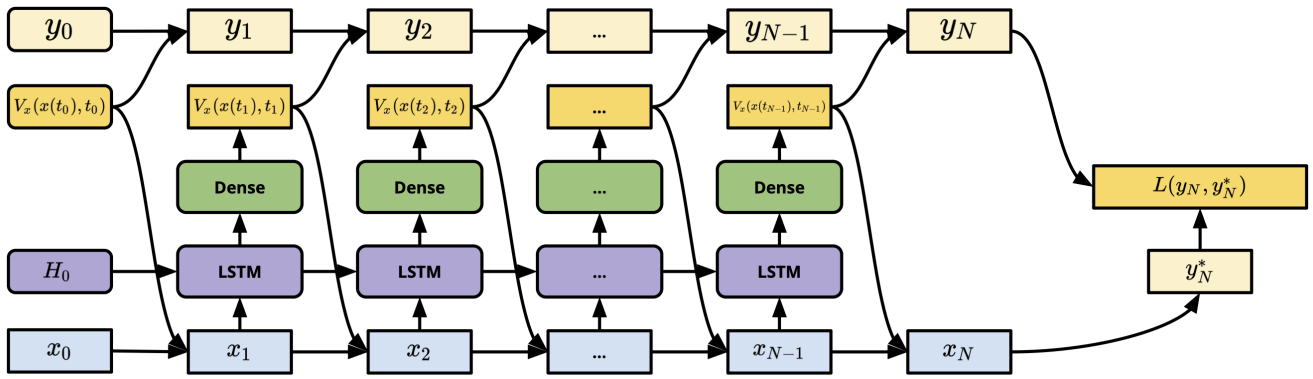

Fig. 2. **Neural Network Architecture**: the architecture here is for N timesteps, the connection between $V_x(x(t_n), t_n)$ and x_{n+1} denotes applying control action, and the weights for each timestep are shared. Curved edge boxes represent trainable parameters, sharp edges represent intermediate values.

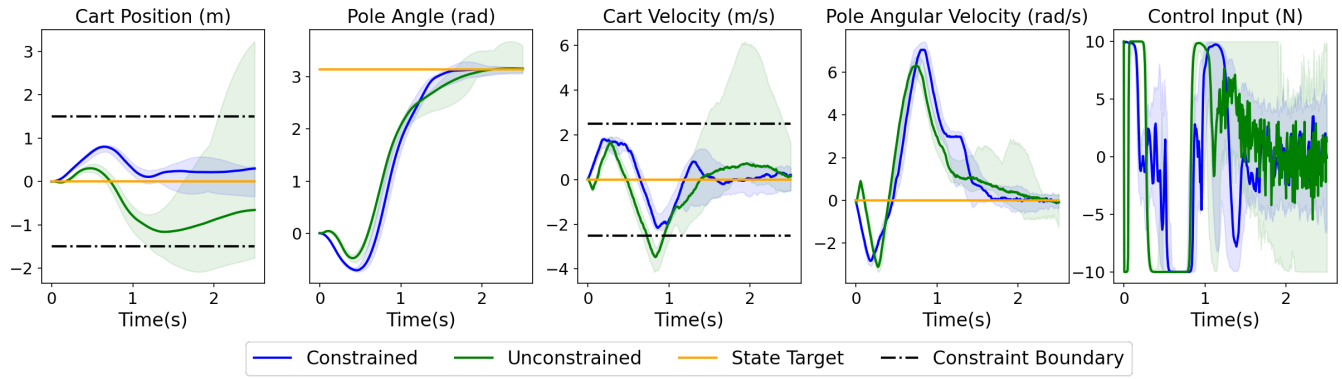

Fig. 3. **Cart-pole State Dynamics**: performance comparison of the controller when the state is constrained (blue) and not constrained (green). The darker blue and green trajectories are sampled trajectories among the 256 trials. The light blue and green regions show the bounds of the maximum and minimum values of all 256 trajectories at each time step. The constraint boundaries are shown in black dashed lines. The orange line shows the target value for each of the states. All the state trajectories of the constrained system satisfy the constraint boundary. Without applying state constraints, the state trajectories (light green region) violate the constraint boundaries.

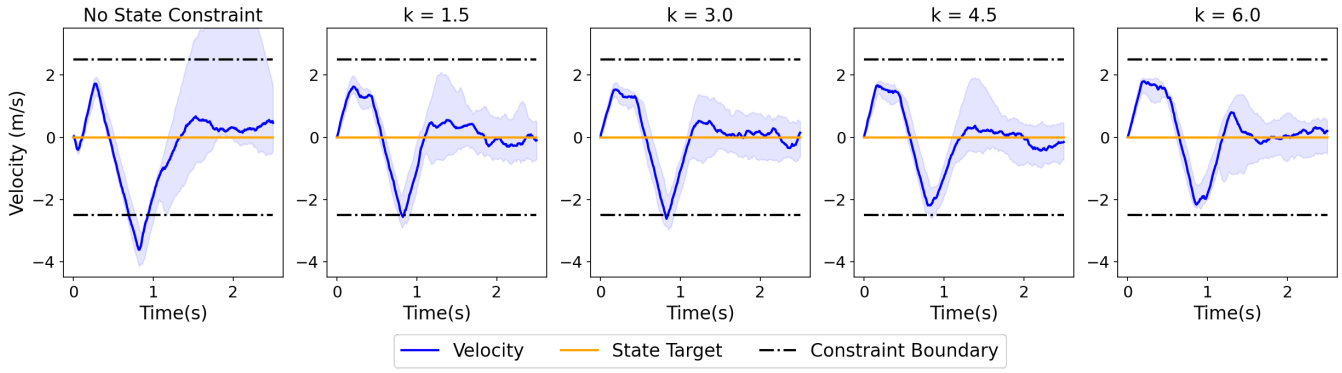

Fig. 4. **Effectiveness of Adaptive Update Scheme**: the color scheme for the velocity follows the constrained curve in Figure 3. The state target and constraint boundary also follows the color scheme in Figure 3. The leftmost figure shows that under no state constraints the cart velocity violates the constraints. After increasing k, the velocity trajectory gradually retreats inside of the boundaries and eventually, it entirely satisfies the constraints.

immediately seen on the state trajectories. With $k = 1.5$, the portion of the trajectory that lies outside of the constraint boundaries is greatly reduced. After gradually increasing k to 6.0 following Algorithm 1, the entire trajectory of the cart velocity lies within the constraint boundaries. Also, we tested directly training on $k = 6.0$ without such gradual increase, the algorithm becomes numerically unstable after one forward pass due to large gradients. This demonstrates that our method provides a stable training scheme.

B. Cart-pole Swing-Up Task II

Using the same model as before, a state constraint is applied to limit the total energy of the system (sum of potential and kinetic energy), which is a nonlinear function

Algorithm 2 State Constrained Deep FBSDE Controller

Given:
$x_0 = \xi$, $f(t,x)$, $G(t,x)$, $\Sigma(t,x)$: Initial state and state dynamics;
$g(x)$, $\nabla_x g(x)$, $q(x)$, R: Cost function parameters;
N: Task horizon, K: Number of iterations, M: Batch size, Δt: Time interval, λ: Weight decay parameter;
Δ, η, η', Δ_δ: state constraint parameters (refer to Algorithm 1);
Initialization:
θ: Weights and biases of dense and LSTM layers;
ϕ: Weight of the initial value function and hidden unit;
k, δ, β, γ (refer to Algorithm 1);
$\{x_0^m\}_{m=1}^M = \{\xi\}_{m=1}^M$;
$\{y_0^m\}_{m=1}^M = \{V(\phi)\}_{m=1}^M$: Value function at $t=0$;
$\{V_x^m(x(t_0),t_0)\}_{m=1}^M = \{V_x(\phi)\}_{m=1}^M$: Gradient of value function at $t=0$;
$\{H_0^m\}_{m=1}^M = \{H_0(\phi)\}_{m=1}^M$: Hidden units at $t=0$;
$t=0$;
for $l=1$ to L **do**
  for $m=1$ to M **do**
    for $n=0$ to $N-1$ **do**
      $u_n^m = -R^{-1}G^T(t,x_n^m)V_x^m(x(t_n),t_n)$
      $u_n^{m*} = U_{\max} * \mathrm{sig}(u_n^m)$;
      Sample Brownian noise: $\Delta w_n^m \sim \mathcal{N}(0,\Delta t)$;
      Update value function:
      $y_{n+1}^m = y_n^m - h(t,x_n^m,y_n^m,V_x^m(x(t_n),t_n))\Delta t + (V_x^m(x(t_n),t_n))^T(G(t,x_n^m)u_n^{m*}\Delta t + \Sigma(t,x_n^m)\Delta w_n^m)$;
      Update system state:
      $x_{n+1}^m = x_n^m + f(t,x_n^m)\Delta t + G(t,x_n^m)u_n^{m*}\Delta t + \Sigma(t,x_n^m)\Delta w_n^m$;
      Predict gradient of value function:
      $V_x^m(x(t_{n+1}),t_{n+1}) = f_{\mathrm{LSTM}}(x_{n+1}^m,\theta^k)$;
    Compute target terminal cost: $(y_N^m)^* = g(x_N^m)$;
  Compute mini-batch loss:
  $L = \frac{1}{M}\sum_{i=1}^{M}\|(y_N^m)^* - y_N^m\|_2^2 + \lambda\|\theta^k\|_2^2$;
  Update θ and ϕ via backpropagation;
  $t = t + \Delta t$;
  Run Algorithm 1;

of the states. For the cart-pole system, the total energy

$$E = \frac{1}{2}M\dot{x}^2 + mgL(1-\cos\theta) + \frac{1}{2}mL^2\dot{\theta}^2 \qquad (23)$$

is constrained at $\pm 5J$. The cost is similar to (22), with the only difference being changing $p(\cdot)$ to incorporate the system energy constraint. The total energy under the unconstrained and constrained controllers are shown in Figure 5.

VII. Conclusion

In this paper, SOC problems with state constraints are formulated as a FBSDE and solved utilizing an LSTM-based DNN to alleviate the curse of dimensionality and numerical integration issues. An adaptive update scheme is used to apply state constraints, which greatly enhanced the stability during training of the LSTM network. The efficacy of our approach is demonstrated on a cart-pole system under two different state constraints: 1) cart position and velocity, and 2) system energy. Potential future directions include application to systems with partial state measurements and robust control under various uncertainties.

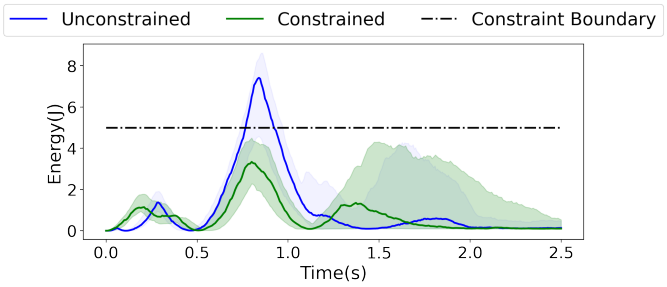

Fig. 5. **Energy Constrained Cart-pole Controller**: the color scheme follows the one from Figure 3. It can be shown that under the constrained controller the total energy is able to be maintained under $5J$ which is not achieved under the unconstrained controller.

References

[1] A. Herdt, N. Perrin, and P. Wieber, "Walking without thinking about it," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, Taipei, Taiwan, October 18-22, 2010*, pp. 190–195.

[2] A. R. Cassandra, L. P. Kaelbling, and J. Kurien, "Acting under uncertainty: discrete Bayesian models for mobile-robot navigation," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, Osaka, Japan, November 4-8, 1996*, pp. 963–972.

[3] P. E. Gill, W. Murray, and M. A. Saunders, "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM Review*, vol. 47, pp. 99–131, 2005.

[4] E. Todorov and Weiwei Li, "A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *Proceedings of the American Control Conference, Portland, OR, June 8-10, 2005*, pp. 300–306.

[5] E. Theodorou, Y. Tassa, and E. Todorov, "Stochastic differential dynamic programming," in *Proceedings of the American Control Conference, Baltimore, MD, June 30 - July 2, 2010*, pp. 1125–1132.

[6] V. A. Huynh, S. Karaman, and E. Frazzoli, "An incremental sampling-based algorithm for stochastic optimal control," *The International Journal of Robotics Research*, vol. 35, no. 4, pp. 305–333, 2016.

[7] I. Exarchos and E. A. Theodorou, "Stochastic optimal control via forward and backward stochastic differential equations and importance sampling," *Automatica*, vol. 87, pp. 159 – 165, 2018.

[8] O. Bokanowski, A. Picarelli, and H. Zidani, "State-constrained stochastic optimal control problems via reachability approach," *SIAM Journal on Control and Optimization*, vol. 54, no. 5, pp. 2568–2593, 2016.

[9] J. Han, A. Jentzen, and W. E, "Solving high-dimensional partial differential equations using deep learning," *Proceedings of the National Academy of Sciences*, vol. 115, no. 34, pp. 8505–8510, 2018.

[10] Z. Wang, M. Pereira, and E. A. Theodorou, "Learning deep stochastic optimal control policies using forward-backward SDEs," in *Proceedings of Robotics: Science and Systems XV, Freiburg im Breisgau, Germany, June 22-26, 2019*.

[11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[13] E. B. Dynkin, "Markov processes: Volume 1," *Springer, Berlin, Heidelberg*, 1965.

[14] R. E. Bellman, "Dynamic programming," *Dover, New York*, 2003.