Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 55-7 (2022) 392–399

Designing neural network control policies under parametric uncertainty: A Koopman operator approach ⋆

Evren M. Turan * **Johannes Jäschke** *

* *Department of Chemical Engineering, Norwegian University of Science and Technology (NTNU), Trondheim, Norway (e-mail: evren.m.turan@ntnu.no, johannes.jaschke@ntnu.no).*

Abstract: Solving the optimisation problem associated with nonlinear model predictive control (MPC) on-line when considering uncertainty is often infeasible due to the large computational times. One approach is to avoid the online optimisation by using an imitation learning approach where a neural network or other approximator is trained on offline solutions of the MPC problem. This can be computationally costly as the potential system behaviour must be fully represented in the data, which requires many MPC solutions.
In this work we propose a new method to train a neural feedback controller in closed loop for problems with uncertainty in parameters and/or the initial conditions. Our method does not require solving MPC problems off-line to generate training data, instead we optimise the neural network directly on the MPC objective in a single shooting approach, with expectations evaluated in their Koopman expectation form using a quadrature algorithm. The proposed method is demonstrated on three problems. The optimised controllers for these problems show good performance, and have an average evaluation time of less than 4 μs.

Keywords: Optimisation under uncertainty, Neural network controller, Optimal control, Explicit MPC

1. INTRODUCTION

Model predictive control (MPC) is a widely used optimisation technique to control a system, given a model. The model takes in the current system state and is used to predict the short term system response, allowing for an optimisation problem to be solved to find the control inputs that minimise some objective, while satisfying constraints. Uncertainty may be addressed in various robust MPC formulations, typically at a significant increase in computational expense. The need to solve the (robust) MPC optimisation problem on-line is a major issue, as this can be infeasible for some processes due to the computational complexity.

In this work we propose to parametrise the control by a neural network, and train this network (off-line) using the expectation of the MPC objective function, and a joint probabilistic constraint, i.e. without solving robust MPC problems to generate training data. This results in a control law that can be cheaply evaluated on-line, while taking into account parametric uncertainty.

Previously it has been proposed to avoid solving the MPC optimisation problem online by pre-computing an explicit control law. After optimisation this control law can then simply be evaluated on-line. For a linear system, without uncertainty, and with a quadratic cost function the optimal control law is known to be a piecewise affine function on polytopes which can be computed in advance (Bemporad et al., 2002). The number of regions defining the piecewise control law rapidly increases with system complexity, thus computing and using this explicit control law can be computationally intractable for large systems (Chen et al., 2018).

Another approach is to define an approximate control law, that is cheap to evaluate and feasible to optimise. As neural networks are universal approximators, and cheap to evaluate after training, numerous authors have proposed the use of a neural network to approximate the control law (a neural control law) with a recent work showing that bounds can be calculated for the neural network size necessary to exactly represent an explicit MPC law (Karg and Lucia, 2020). Neural control laws have been optimised by 1) using MPC solutions evaluated at various points (imitation learning) (Hertneck et al., 2018; Karg and Lucia, 2020), 2) reinforcement learning, 3) using a neural differential equation approach where the loss is the MPC objective (Rackauckas et al., 2020; Sandoval et al., 2021) or 4) a mixture of approaches, e.g. (Karg and Lucia, 2021).

The imitation learning (1) approach can be very costly, as the data set containing MPC solutions needs to fully describe the system behaviour, which requires solving an MPC problem many times, although this can be done off-line. We note that there are various approaches to generate the data for imitation learning, including algorithms to generate "rich" training data, to aid in the scaling of the

⋆ This research was conducted as part of the AutoPRO project funded by the Norwegian research council.

10.1016/j.ifacol.2022.07.475

method to higher dimensions (Bonzanini et al., 2021). An important advantage of imitation learning is that one can approximate a robust (stationary) controller, e.g. a controller robust to input disturbances (Hertneck et al., 2018) or a multi-stage NMPC (Karg and Lucia, 2021; Bonzanini et al., 2021).

In the reinforcement learning approach (2) the neural network parameters are updated on-line based on the state of the system and its response to control outputs. A variation of this approach is to optimise the network off-line using a system model (Chen et al., 2018). This is similar to training the network in a neural differential equation approach (3), where instead of a loss defined by data, the MPC objective function is used at discrete time points (Rackauckas et al., 2020; Sandoval et al., 2021). In the control literature, this later approach is essentially an indirect, single shooting method (Biegler, 2010).

In this work, we build on approach (3) to train a robust neural controller, for problems with parametric uncertainties, without solving a robust MPC problem. We train the neural network via single shooting, where the expectation of the constraint and objectives are evaluated in their Koopman expectation form, using a quadrature algorithm.

2. BACKGROUND

2.1 Nonlinear model predictive control

Consider the general nonlinear dynamic system,

$$\dot{x}(t) = f(x(t), u(t), p) \tag{1}$$

where t is time, $x \in \mathbb{R}^{n_x}$ are the states, $u \in \mathbb{R}^{n_u}$ are the control inputs, $p \in \mathbb{R}^{n_p}$ are parameters, and f describes the dynamics.

The goal of nonlinear model predictive control (NMPC) is to compute the optimal state and control trajectories ($\mathbf{x} = [x_1, \ldots, x_{N_k}]$ and $\mathbf{u} = [u_0, \ldots, u_{N_{k-1}}]$) for a given prediction horizon. NMPC requires the initial state of the system, which we assume is available. Typically, the MPC problem takes the discretised form:

$$\min_{\mathbf{x},\mathbf{u}} J = \min_{\mathbf{x},\mathbf{u}} \sum_{k=0}^{N_k-1} l(x_k, u_k, p, k) + V(x_{N_k}) \tag{2a}$$

$$x_{k+1} = f_d(x_k, u_k, p), \quad \forall k = 0, \ldots, N_k - 1 \tag{2b}$$

$$0 \geq g(x_k, u_k, p), \quad \forall k = 1, \ldots, N_k \tag{2c}$$

$$x_{lbd} \leq x_k \leq x_{ubd}, \quad \forall k = 1, \ldots, N_k \tag{2d}$$

$$u_{lbd} \leq u_k \leq u_{ubd}, \quad \forall k = 0, \ldots, N_k \tag{2e}$$

$$x_{k=0} = x_{init} \tag{2f}$$

where J is the objective function, N_k the prediction horizon points, l the stage cost, V the terminal cost, f_d the discretised dynamics, g a vector of nonlinear constraints, and x_{init} denotes the initial conditions. Bounds (x_{lbd}, x_{ubd}, u_{lbd}, u_{ubd}) are defined for the state and controls. In this formulation constraints are enforced at the discrete time points.

2.2 NMPC with parametric uncertainty

Consider the NMPC problem, but where p and/or x_{init} are uncertain, and described by some probability density functions. We assume that the uncertain parameters are not time varying. Thus, uncertainty in the parameters and initial conditions are equivalent, i.e. if we augment the system equations with the parameter vector (with no dynamics) then there will only be uncertainty in the initial states.

Given probability density functions for the parameters π_p and initial state $\pi_{x_{init}}$ we can write the stochastic MPC problem, where the objective is given as an expectation:

$$\min_{\mathbf{x},\mathbf{u}} \quad \mathbb{E}_{p,x_{k=0}}\left[J(\mathbf{x}, \mathbf{u}, p)\right], \qquad p \sim \pi_p, \; x_{k=0} \sim \pi_{x_{init}} \tag{3}$$

If desirable we could also include the variance of the cost $\mathbb{V}[J]$ in the optimisation, noting that $\mathbb{V}[J] = \mathbb{E}[(J - \mathbb{E}[J])^2]$.

We define a joint chance constraint that the probability of g, the vector function of constraints, being satisfied at all time points of the discretisation is greater than $1 - \epsilon$:

$$\mathbb{P}_{p,x_{k=0}}\left(\bigcap_{k=0}^{N_k} g(x_k, u_k, p) \leq 0\right) \geq 1 - \epsilon, \qquad 0 \leq \epsilon \leq 1 \tag{4}$$

where $\mathbb{P}$ is the probability. If possible, specifying $\epsilon = 0$ can significantly increase the cost (Eq. 3) as it is highly conservative. The chance constraint can be written as an expectation by defining an indicator function (I):

$$I(x_k, u_k, p) = \begin{cases} 1 & g(x_k, u_k, p) \leq 0 \\ 0 & \text{else} \end{cases} \tag{5}$$

$$\mathbb{E}_{p,x_{k=0}}[I(x_k, u_k, p)] = \mathbb{P}\left(\bigcap_{k=0}^{N_k} g(x_k, u_k, p) \leq 0\right) \tag{6}$$

Ignoring the uncertainty and picking a nominal p can lead to a controller that performs poorly, despite feedback from the system (Kothare et al., 1996). Solving the original NMPC problem (2) can be difficult, and under uncertainty the problem becomes even more computationally expensive. Various formulations have been proposed for nonlinear and linear MPC under uncertainty including: solving a worst-case objective function (Kothare et al., 1996), multi-stage MPC (Lucia et al., 2017) and tube-based MPC (Mayne et al., 2011). In general, treating the uncertainty in an online optimisation (as in multi-stage MPC) is very costly. On the other hand this computational cost can be moved off-line by training a neural network, which can lead to practically the same performance with negligible on-line computational cost.

3. OPTIMISATION OF THE NEURAL NETWORK CONTROL POLICY

3.1 Overview of the method

We combine the closed loop simulation of the neural network policy, the evaluation of the expectation, and the optimisation as shown in Figure 1. At each iteration of the method we take neural network parameters θ_k and evaluate the expectation of the (penalised) objective. This expectation is evaluated in an inner loop wherein we adaptively sample p and $x_{k=0}$ from $\pi_{x_{init}}$ and π_p. For each sample we perform a closed loop simulation of the control policy, in which the value of the constraints and cost function are evaluated. The adaptive sampling is performed until the expectation is evaluated to a specified tolerance.

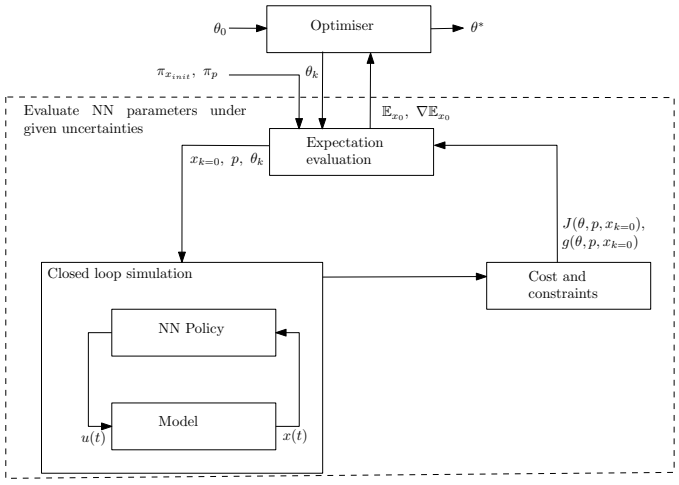


Fig. 1. Block diagram of the training process of the neural network control policy.

The main features of the workflow are that 1) the network is optimised via closed loop simulations and 2) that the network is optimised on the expectations of the system, which are evaluated by "pulling back" the uncertainty by applying the Koopman operator. These two steps are described in detail below.

3.2 Closed loop simulation of the control policy

To avoid solving an NMPC problem with uncertainty in real time, we focus on optimising an off-line control law given by a neural network (NN), with parameters θ: $u(x) = f_{NN}(x|\theta)$, i.e. a neural network policy. Neural networks can be used in to approximate the control law as from the universal approximation theorem, they can approximate any function to a desired tolerance, under mild conditions. Historically, Parisini and Zoppoli (1995) first proposed the use of neural networks to approximate an MPC law. More recently, Karg and Lucia (2020) showed that neural networks can exactly represent the explicit MPC law for linear time-invariant systems.

There are various network architectures, and in this work we chose to use deep, fully connected, feedforward neural networks. Such a network has the form:

$$f_{NN}(x|\theta) = \beta_{L+1} \circ f_{L+1} \circ \beta_L \circ f_L \ldots \beta_1 \circ f_1(x) \tag{7}$$

where L is the number of hidden layers. Each hidden layer, l, consists of a nonlinear activation function, g_l and an affine function, f_l, given by:

$$f_l = W_l \zeta_{l-1} + b_l \tag{8}$$

where W_l and b_l are the weights and biases of the affine function, and ζ_{l-1} is the output from the previous layer ($\zeta_0 = x$). Common choices for β_l include rectified linear units, sigmoid functions, and hyperbolic tangent functions. For convenience we define θ to be a vector containing the weights and biases, i.e. $\theta = [W_1,\ b_1,\ \ldots,\ W_L,\ b_L]$.

To explain how the network parameters are optimised, consider a dynamical system with no uncertainty (Equation 1). Given a neural network policy, this means the dynamics are given by the neural differential equation (Rackauckas et al., 2019):

$$\dot{x}(t) = f(x(t), f_{NN}(\theta), p) \tag{9}$$

Consider the deterministic MPC objective with only a constraint to enforce the dynamics (equations 2a-b). Given θ, p, and the initial conditions, x_{init}, we can find the sequence x_k and u_k simply by integrating the neural differential equation (i.e. the block "Closed loop simulation" in Figure 1) which enforces the constraint 2b. After integration, the sequence x_k and u_k can then be used to evaluate the objective function (equation 2a).

The gradient with respect to network parameters is given by the chain rule, e.g. $\frac{dJ}{d\theta} = \frac{dJ}{dx_k}\frac{dx_k}{d\theta}$, and can be evaluated via automatic differentiation on the integrator. This allows the use of any gradient based optimiser for finding the network parameters.

In the control literature this formulation of the optimisation problem is essentially a single shooting, optimal control problem (Biegler, 2010), where the manipulated variables are the unconstrained network parameters. The suggested parametrisation of the control policy, $u(x) = f_{NN}(x|\theta)$, is known as a state feed-back neural policy in the reinforcement learning literature. However, here the optimisation is performed entirely off-line as in Sandoval et al. (2021). We note that this approach does have difficulties with unstable systems, however first training the network on a nominal control profile and then proceeding with the optimisation is a potential approach to overcome this (Sandoval et al., 2021).

In this approach the control law is optimised in a closed loop manner. Once optimised, the network defines a control law that is a function of the current state and in online operation can simply be evaluated, similar to an explicit MPC. In comparison to an explicit MPC, the memory requirements for a neural network control law can be significantly smaller (Karg and Lucia, 2020).

3.3 Expectation evaluation

Assume we have a control law $u = f_{NN}(x|\theta)$, and that uncertainties in the parameters have been transformed to uncertainties in the initial state, i.e. we define

$$x_0 = \begin{bmatrix} x_{init} \\ p \end{bmatrix} \tag{10}$$

The Koopman operator provides a way to evaluate the expectations that occur in optimisation under uncertainty (equation 3 and 4) in a relatively efficient manner (Gerlach et al., 2020). The main idea is that if the dynamics of a system are deterministic, then the probability of a distribution in the future is solely dependent on the initial distribution, i.e. the uncertainty can be "pulled back".

Let x_0 be the uncertain initial state, described by a valid probability density function π_0, and x_T be the uncertain state at some future time T, described by the probability distribution function π_T. Let $h(x)$ be some real valued function of the state space, (e.g. the constraints 2c-2e), that we wish to evaluate, and S be the mapping of the system from $t = 0$ to some time T, $S : \mathbb{R}^{n_x} \to \mathbb{R}^{n_x}$. Assuming that S is measurable and non-singular, and that h is continuously differentiable and has compact support, then the following are equivalent (Lasota and Mackey, 1994):

$$\mathbb{E}_{x_T}[h(x_T)|x_T \sim \pi_T] = \mathbb{E}_{x_0}[h(S(x_0))|x_0 \sim \pi_0] \quad (11)$$

$$= \int_\Omega h(S(x))\pi_0(x)\mathrm{d}x \quad (12)$$

where Ω is the state space. The right-hand side of equation 11 is known as the Koopman expectation, which is explicitly written as an integral in equation 12. Equation 11 uses the adjoint property of the Koopman and Frobenius-Perron operators. For more information on these operators we direct the reader to Lasota and Mackey (1994) (Chapter 1 and 3) or Leonard (2019), and the references therein, for a focus on the Koopman operator in optimisation problems.

Equation 11 states that determining the expectation at time T, given the corresponding probability distribution (π_T), is equivalent to using the initial probability distribution (π_0) and the system mapping. Note that the mapping does not have to be known, only its evaluation. This is significant as we typically know the system uncertainty at its initial state.

To evaluate $\mathbb{E}[h(x_T)|x_T \sim \pi_T]$ we could take samples from the initial distribution, π_0, and push them through the system dynamics, forming a Monte Carlo estimate of π_T. Using the estimate of π_T one could then evaluate $\mathbb{E}[h(x_T)|x_T \sim \pi_T]$. Forming this estimate of π_T is typically computationally expensive as many samples are needed.

However, there are various benefits to evaluating the Koopman expectation form (Eq. 12) instead (Meyers et al., 2019; Gerlach et al., 2020). As we know π_0, and can evaluate $h(S(x))$, the integral can be evaluated by any numerical integration technique, e.g. multidimensional quadrature, (quasi-) Monte Carlo integration. If an adaptive integration procedure is used, then realisations of the uncertain parameters will be selected to evaluate $\mathbb{E}[h(S(x_0))|x_0 \sim \pi_0]$ to a desired tolerance (Gerlach et al., 2020). In other words, instead of fixing the location or number of samples we specify the desired tolerance directly.

In this work the integration is performed via a multidimensional h-adaptive quadrature (Genz and Malik, 1980). In this type of quadrature the integral is evaluated by adaptively subdividing the integration volume into smaller volumes, while applying the same fixed-order quadrature rule within each sub-region. Quadrature can be very efficient in low dimensions (here the number of uncertain parameters and states), however it is known to scale exponentially with the number of dimensions. An alternative choice for high dimensional problems are adaptive quasi-Monte Carlo integration algorithms, e.g. see the CUBA library (Hahn, 2005). Note that the point at which quasi-Monte Carlo algorithms are more efficient that quadrature algorithms depends on both the dimensionality and the behaviour of the integrand – see Schürer (2003) for a comparison.

Regardless, by evaluating the expectation in the Koopman form the computational costs can be significantly reduced compared to forming the Monte Carlo estimate of π_T (Meyers et al., 2019; Gerlach et al., 2020). Using this approach the overall cost of performing optimal control under uncertainty can be significantly reduced (Meyers et al., 2019).

3.4 Remarks on the Implementation

As discussed, uncertainty in the parameters and/or initial state can be considered, by evaluating the expectation of the cost (equation 3) using Eq. 11. We note that π_{x_0} could describe both the uncertainty in initial conditions of a batch process, or the operating region of the state space (e.g. estimated from data). In both of these cases π_{x_0} could be a complex multi-dimensional probability distribution, especially when there are constraints on the states. It should be noted that the controller is not trained only on state space given by π_{x_0}, i.e. the closed loop optimisation means that the controller will be receive state values, based on the controller outputs. Due to this approach, important regions of the space will be heavily sampled if the horizon used in the controller optimisation is long enough.

When performing the optimisation, box constraints on the input u can be treated in the network design, e.g. use a sigmoid activation on the final layer to limit the output between 0 and 1. To enforce state constraints we use a penalty approach, where we define the penalised objective ϕ:

$$\min_\theta \mathbb{E}_{x_0}[\phi(\theta, x_0)] = \min_\theta \mathbb{E}_{x_0}[J(\theta, x_0) + \rho Q(I(\theta, x_0))] \quad (13)$$

where I is the indicator function, ρ is an adjustable penalty parameter and Q is a penalty function. We remind that x_0 is the augmented vector with parameters as states.

Common choices for Q include the l_1 and l_∞ norms, and the quadratic penalty function, $Q(x) = x^2$. Using the norms gives an exact penalty function, which means that minimization with some ρ^* will give the same solution as the constrained problem, under some assumptions (Biegler, 2010). One may then iteratively evaluate $\mathbb{E}[I(\theta, p, x_0)]$ and increase ρ until the desired satisfaction is met. Although the quadratic penalty is not exact it is still commonly used for numerical convenience.

An alternative choice that is also not equivalent to the constrained optimisation is to penalise the extent of constraint violation, e.g. using a quadratic penalty:

$$Q(g) = \sum_{i=0}^{N_g} \sum_{k=0}^{N_k} (\max(g_i(x_k, u_k, p), 0.0))^2 \quad (14)$$

Ultimately, the choice of penalty function depends on the system considerations that apply.

In summary, we perform gradient-based optimisation to find the parameters of a neural network feedback policy for a system with uncertainty in the parameters and/or initial conditions, using the expectation of the penalised cost $\mathbb{E}[\phi]$, as the objective function, evaluated in the Koopman expectation form via quadrature. As is standard, hyperparameters of the formulation, e.g. network layer width, can be set by Bayesian optimisation or similar by performing the proposed workflow in an "inner-loop".

An alternative approach is to estimate $\mathbb{E}[\phi]$ using a small number of samples, and use a stochastic optimisation algorithm. The advantage of taking a small number of samples is that the expectation evaluations will be less costly. The estimate of the expectation may be inaccurate due to the number of samples, however if it is unbiased

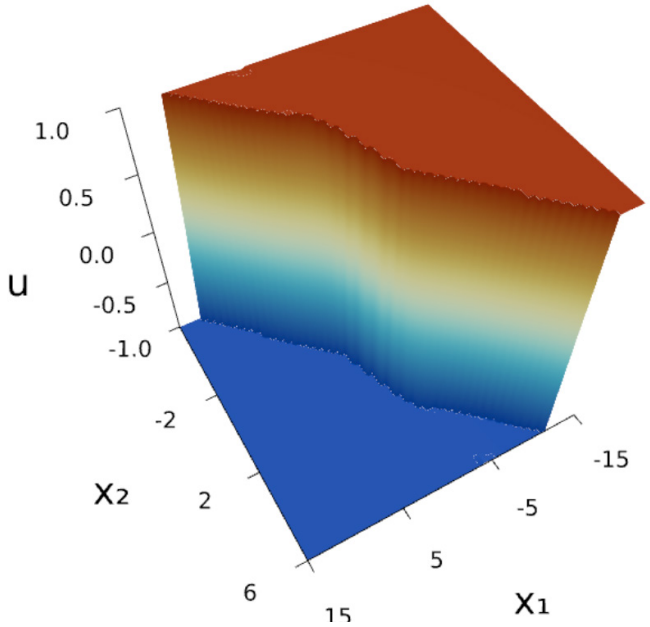


Fig. 2. Neural network control for the double integrator system

then the stochastic optimiser will converge. Thus, such an approach will trade off faster iterations, due to the low number of samples, with slower convergence of the optimiser (Bottou and Bousquet, 2012). We note that it would be feasible to begin with a stochastic optimisation approach and then switch to using the proposed approach.

The proposed workflow is coded in Julia (Bezanson et al., 2017), and applied to three case studies, using the following packages: DifferentialEquations (Rackauckas and Nie, 2017), DiffEqFlux (Rackauckas et al., 2019),DiffEqUncertainty (Gerlach et al., 2020), Flux (Innes et al., 2018), ForwardDiff (Revels et al., 2016), HCubature, and NLopt (Johnson, 2014). We note that this combination of packages supports the automatic differentiation of equation 13. The neural networks are initialised with Glorot initialisation (Glorot and Bengio, 2010). The case studies were performed on an Intel i5-10310U CPU.

4. CASE STUDIES

4.1 Double integrator

Consider the double integrator, with uniform uncertainty in the initial condition, $x_{init} = [\mathcal{U}(-15, 15), \mathcal{U}(-6, 6)]$, and no parametric uncertainty:

$$\frac{dx_1}{dt} = x_2 \tag{15}$$

$$\frac{dx_2}{dt} = u \tag{16}$$

The objective is to control the system to the origin, with constraints $-1 \leq u \leq 1$, and stage cost:

$$l(x, u, t) = x(t)^2 + 0.1u(t) \tag{17}$$

One can find the explicit MPC for this system, as in Bemporad et al. (2002), using the ranges $x_1 \in [-15, 15]$ and $x_2 \in [-6, 6]$.

The control horizon used for optimising the neural network is 5 seconds, with the objective calculated at time points

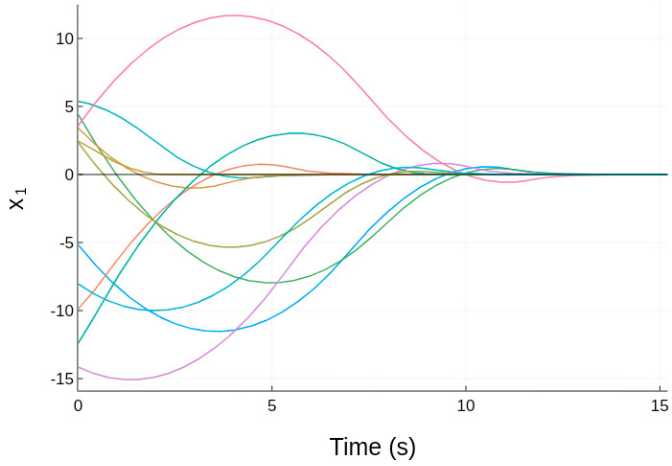


Fig. 3. Multiple state trajectory of the double integrator system from different initial conditions, controlled by the trained neural network.

of 1 second intervals. The network has 2 hidden layers of 10 nodes each, without bias nodes, tanh as the activation functions and the control constraints are satisfied by a "hard tanh". Optimisation is performed with LBFGS (Liu and Nocedal, 1989).

Quadrature for the expectation is performed with an absolute tolerance 10^{-3}. The trained neural network control law is shown in Figure 2. Using the neural controller, the expected cost over a 40 second period is 851 while for an explicit MPC (with 7 regions) it is 873. Examples of the controlled trajectory of x_1 is shown in Figure 3. Note that the control is mostly at its constraints which is responsible for the slow response seen in Figure 3. Performance is improved by increasing the bound on the control.

After training the mean and median execution time of the neural network is 2.576 μs (10^{-6}s) and 2.171 μs respectively.

4.2 Linear angular positioning system

This is a linear model of an angular positioning system described in Kothare et al. (1996):

$$\frac{d\theta_1}{dt} = \theta_2 \tag{18}$$

$$\frac{d\theta_2}{dt} = \alpha\theta_2 + \kappa u \tag{19}$$

$$|u| \leq 2 \tag{20}$$

where θ are the states (rad), $\kappa = 0.787$ (rad^{-1}V^{-1}s^{-2}) and α (s^{-1}) is an uncertain parameter, assumed to be uniformly distributed between 0.1 and 10. The objective is to return an initially disturbed state to the origin, with the stage cost:

$$l(\theta, u, t) = \theta_1(t)^2 + 10^{-5}u(t)^2 \tag{21}$$

To parametrise the controller we use a neural network with three layers of eight neurons each, and no bias nodes. Rectified linear units are used as activation functions, and the constraint on the control is enforced by a "hard tanh". The network is trained on a 4 second time range, with the objective evaluated at 0.1 intervals, and the initial point fixed to $x_{init} = [0.05, 0.0]$. Optimisation is performed with

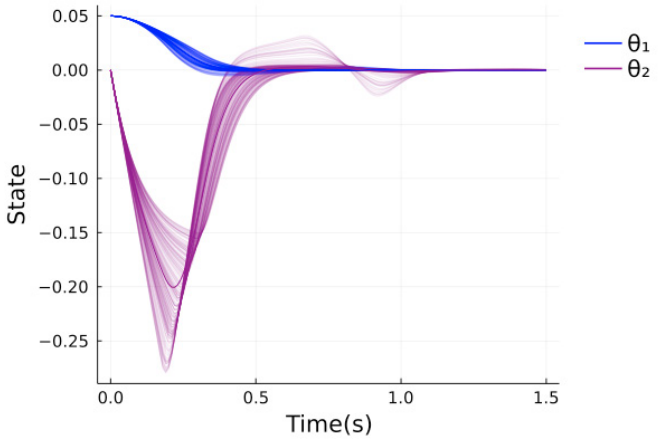


Fig. 4. State trajectories of the controlled angular positioning system, with parameter α drawn from its distribution.

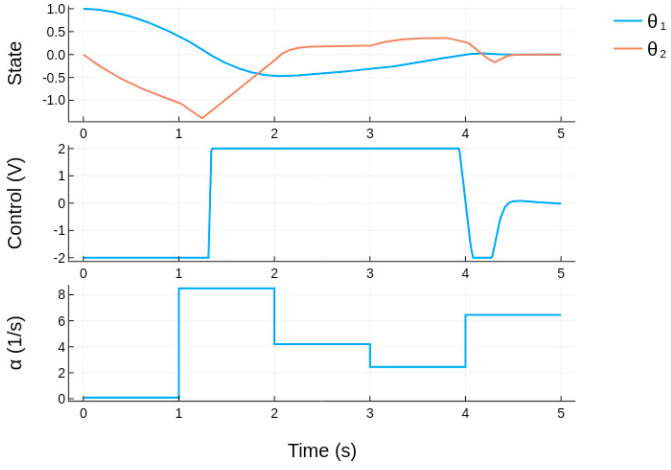


Fig. 5. Profile of θ_1 and control output for the angular positioning system with randomly time varying α

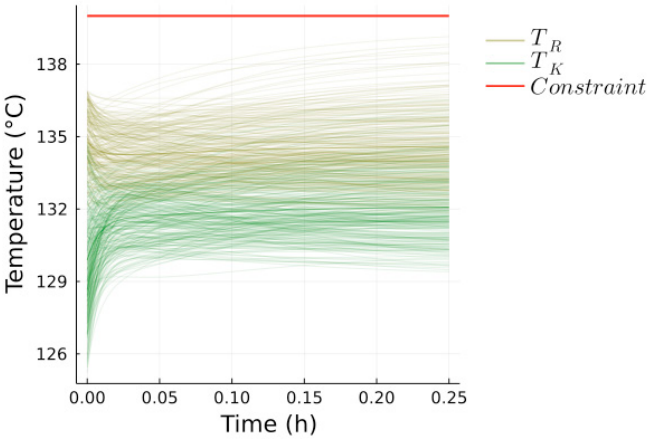
LBFGS (Liu and Nocedal, 1989). Quadrature is performed with an absolute tolerance of 10^{-5}.

The state profile of the trained system is shown in Figure 4. In this plot each trajectory corresponds to a closed loop simulation of the system with a different realisation of the parametric uncertainty. Figure 5 shows a closed-loop simulation with the uncertain parameter varying in time.

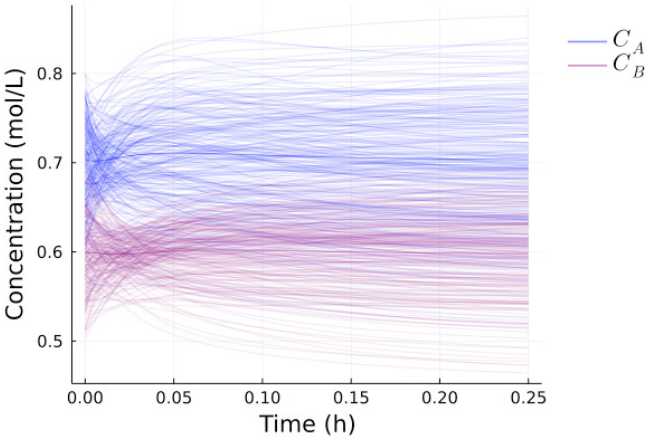
After training the mean and median execution time of the neural network is 3.620 μs and 3.188 μs respectively, with an estimated memory requirement of 8.20 KiB.

4.3 Nonlinear CSTR

The proposed approach is demonstrated on a nonlinear continuous stirred tank reactor (CSTR) based on the implementation of Lucia et al. (2017), with uncertainty in the states and parameters. The CSTR model is given by the following equations:



(a)



(b)

Fig. 6. State trajectories of the controlled CSTR, using parameters drawn from their distributions, after optimisation of the neural controller. Temperature constraint marked in red, in (a). Constraint satisfaction is 100%.

$$\frac{dC_A}{dt} = F(C_{A,0} - C_A) - k_1 C_A - k_3 C_A^2 \tag{22a}$$

$$\frac{dC_B}{dt} = FC_B + k_1 C_A - k_2 C_B \tag{22b}$$

$$\frac{dT_R}{dt} = \frac{k_1 C_A H_{R,1} + k2 C_B H_{R,2} + k_3 C_A^2 H_{R,3}}{-\rho c_P} + F(T_{in} - T_R) + \frac{K_w A_R (T_R - T_K)}{\rho c_P V_R} \tag{22c}$$

$$\frac{dT_K}{dt} = \frac{Q + k_w A_R (T_R - T_K)}{m_k C_{P,k}} \tag{22d}$$

with kinetic expression:

$$k_1 = \beta k_{0,1} \exp\left(\frac{-E_{A,1}}{T_R + 273.15}\right) \tag{23}$$

$$k_2 = k_{0,2} \exp\left(\frac{-E_{A,2}}{T_R + 273.15}\right) \tag{24}$$

$$k_3 = k_{0,3} \exp\left(\frac{-\alpha E_{A,3}}{T_R + 273.15}\right) \tag{25}$$

where parameters α and β are normally distributed, $\mathcal{N}(1., 0.02)$ and $\mathcal{N}(1., 0.05)$, with the distributions truncated at 1 ± 0.05 and 1 ± 0.1 respectively. The four states

are the concentrations of A and B (C_A, C_B, mol/L), with bounds [0.1, 2.], and the temperatures of the reactor and cooling jacket (T_R, T_K, °C) with bounds [50, 140]. The initial condition is given by independent normal distributions: $\pi_{x_0} = [\mathcal{N}(0.7, 0.05),\ \mathcal{N}(0.6, 0.05),\ \mathcal{N}(135, 1.5),\ \mathcal{N}(130, 2.5)]$. These distributions are truncated at the mean ±0.1 mol/L for the concentrations, ±3°C for T_R, and ±5°C for T_K.

The feed (F, L/h) and heat flow (Q, kW) are control inputs, with bounds of [5., 100.] and [−8500, 0.] respectively. Other parameter values are listed in Lucia et al. (2017). Note that in the above time (t) is in hours. We consider the stage cost:

$$l = (C_B(t) - 0.6)^2 + 0.1(C_A(t) - 0.706)^2 + 0.001\left((T_R(t) - 135)^2 + (T_K(t) - 133)^2\right) \tag{26}$$

As the objective function we use the expectation of the penalised cost plus two times the variance, i.e.

$$\min_{\theta} \mathbb{E}_{p,x_0}[\phi] + 2\mathbb{V}_{p,x_0}[\phi] \tag{27}$$

where the penalised cost is defined as in Eq. 13.

A neural network with two hidden layers of six nodes each is used, with tanh as the activation functions, and with control constraint satisfaction enforced by a sigmoid function. We use normalised states and control outputs for the network due to the differences in magnitude in these variables. The network is trained using a 0.1875 hours (11.25 minutes) time interval, with data recorded every 0.025 hours (1.5 minutes).

Optimisation is performed by LBFGS (Liu and Nocedal, 1989). Quadrature is performed with an absolute tolerance of 10^{-4}. We use a penalty approach, aiming for a joint chance constraint of 100% satisfaction.

250 trajectories of the controlled system are shown in Figure 6. The controller is able to stabilise the states, with 100% constraint satisfaction. State profiles of the controlled system with randomly varying parameters is shown in Figure 7. Despite the varying parameters the controller is able to keep the states near their set points.

After training the mean and median execution time of the neural network is 1.91 μs and 1.86 μs respectively, with an estimated memory requirement of 3.86 KiB.

5. CONCLUSION

We have shown that neural networks can be optimised to give an explicit control law for systems with uncertainty in the parameters and/or initial state. The optimisation was performed with expectations evaluated in their Koopman expectation form, for the numerical benefits described in (Meyers et al., 2019; Gerlach et al., 2020).

In comparison to an explicit MPC law of 7 regions for double integrator example, the neural policy gives a similar solution. The use of a neural policy to ensure near complete probabilistic constraint satisfaction is shown on a nonlinear example. The use of neural networks in this context is interesting as after training they are fast to evaluate, with all evaluation times in this paper being

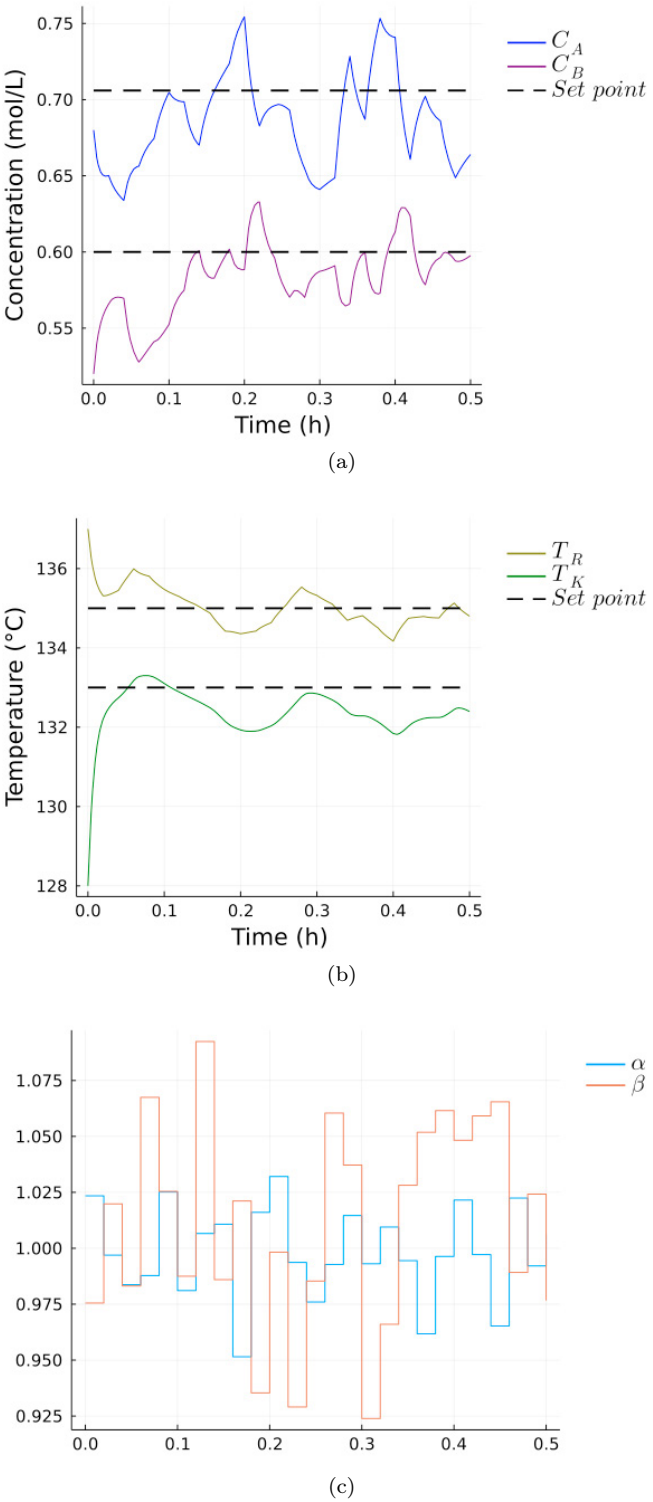


Fig. 7. State trajectories of the controlled CSTR, with randomly varying parameters, after optimisation of the neural controller are shown in (a) and (b). The parameter were randomly drawn from their respective distributions at 0.02 h intervals as shown in (c).

less than 4 μs. Future work could include a comparison to other training approaches, such as imitation learning, and a comparison of different approaches to evaluating the expectations, e.g. polynomial chaos expansion and the unscented transformation.

REFERENCES

Bemporad, A., Morari, M., Dua, V., and Pistikopoulos, E.N. (2002). The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1), 3–20. doi:10.1016/S0005-1098(01)00174-1.

Bezanson, J., Edelman, A., Karpinski, S., and Shah, V.B. (2017). Julia: A fresh approach to numerical computing. doi:10.1137/141000671. URL `https://epubs.siam.org/doi/10.1137/141000671`.

Biegler, L.T. (2010). *Nonlinear programming: concepts, algorithms, and applications to chemical processes*. Society for Industrial and Applied Mathematics.

Bonzanini, A.D., Paulson, J.A., Makrygiorgos, G., and Mesbah, A. (2021). Fast approximate learning-based multistage nonlinear model predictive control using Gaussian processes and deep neural networks. *Computers and Chemical Engineering*, 145, 107174. doi:10.1016/j.compchemeng.2020.107174.

Bottou, L. and Bousquet, O. (2012). *The Tradeoffs of Large-Scale Learning*, 351–368. The MIT Press.

Chen, S., Saulnier, K., Atanasov, N., Lee, D.D., Kumar, V., Pappas, G.J., and Morari, M. (2018). Approximating Explicit Model Predictive Control Using Constrained Neural Networks. *Proceedings of the American Control Conference*, 2018-June, 1520–1527. doi:10.23919/ACC.2018.8431275.

Genz, A. and Malik, A. (1980). Remarks on algorithm 006: An adaptive algorithm for numerical integration over an N-dimensional rectangular region. *Journal of Computational and Applied Mathematics*, 6(4), 295–302. doi:10.1016/0771-050X(80)90039-X.

Gerlach, A.R., Leonard, A., Rogers, J., and Rackauckas, C. (2020). The Koopman Expectation: An Operator Theoretic Method for Efficient Analysis and Optimization of Uncertain Hybrid Dynamical Systems. 1–18. URL `http://arxiv.org/abs/2008.08737`.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 249–256.

Hahn, T. (2005). Cuba—a library for multidimensional numerical integration. *Computer Physics Communications*, 168(2), 78–95. doi:10.1016/j.cpc.2005.01.010.

Hertneck, M., Kohler, J., Trimpe, S., and Allgower, F. (2018). Learning an Approximate Model Predictive Controller with Guarantees. *IEEE Control Systems Letters*, 2(3), 543–548. doi:10.1109/LCSYS.2018.2843682.

Innes, M., Saba, E., Fischer, K., Gandhi, D., Rudilosso, M.C., Joy, N.M., Karmali, T., Pal, A., and Shah, V. (2018). Fashionable Modelling with Flux. *CoRR*, abs/1811.0. URL `https://arxiv.org/abs/1811.01457`.

Johnson, S.G. (2014). The NLopt nonlinear-optimization package. URL `http://github.com/stevengj/nlopt`.

Karg, B. and Lucia, S. (2020). Efficient Representation and Approximation of Model Predictive Control Laws via Deep Learning. *IEEE Transactions on Cybernetics*, 50(9), 3866–3878. doi:10.1109/TCYB.2020.2999556.

Karg, B. and Lucia, S. (2021). Reinforced approximate robust nonlinear model predictive control. In *2021 23rd International Conference on Process Control (PC)*, 149–156. IEEE. doi:10.1109/PC52310.2021.9447448.

Kothare, M.V., Balakrishnan, V., and Morari, M. (1996). Robust constrained model predictive control using linear matrix inequalities. *Automatica*, 32(10), 1361–1379. doi:10.1016/0005-1098(96)00063-5.

Lasota, A. and Mackey, M.C. (1994). *Chaos, Fractals, and Noise*, volume 97 of *Applied Mathematical Sciences*. Springer New York, New York, NY. doi:10.1007/978-1-4612-4286-4.

Leonard, A. (2019). *Probabilistic Methods for Decision Making in Precision Airdrop*. Ph.D. thesis, Georgia Institute of Technology.

Liu, D.C. and Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1-3), 503–528. doi:10.1007/BF01589116.

Lucia, S., Tătulea-Codrean, A., Schoppmeyer, C., and Engell, S. (2017). Rapid development of modular and sustainable nonlinear model predictive control solutions. *Control Engineering Practice*, 60, 51–62. doi:10.1016/j.conengprac.2016.12.009.

Mayne, D.Q., Kerrigan, E.C., van Wyk, E.J., and Falugi, P. (2011). Tube-based robust nonlinear model predictive control. *International Journal of Robust and Nonlinear Control*, 21(11), 1341–1353. doi:10.1002/rnc.1758.

Meyers, J.J., Leonard, A.M., Rogers, J.D., and Gerlach, A.R. (2019). Koopman Operator Approach to Optimal Control Selection Under Uncertainty. In *2019 American Control Conference (ACC)*, 2964–2971. IEEE. doi:10.23919/ACC.2019.8814461.

Parisini, T. and Zoppoli, R. (1995). A receding-horizon regulator for nonlinear systems and a neural approximation. *Automatica*, 31(10), 1443–1451. doi:10.1016/0005-1098(95)00044-W.

Rackauckas, C., Innes, M., Ma, Y., Bettencourt, J., White, L., and Dixit, V. (2019). Diffeqflux.jl-A julia library for neural differential equations. *arXiv preprint arXiv:1902.02376*.

Rackauckas, C., Ma, Y., Martensen, J., Warner, C., Zubov, K., Supekar, R., Skinner, D., Ramadhan, A., and Edelman, A. (2020). Universal Differential Equations for Scientific Machine Learning. URL `http://arxiv.org/abs/2001.04385`.

Rackauckas, C. and Nie, Q. (2017). Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia. *Journal of Open Research Software*, 5(1).

Revels, J., Lubin, M., and Papamarkou, T. (2016). Forward-Mode Automatic Differentiation in Julia. URL `http://arxiv.org/abs/1607.07892`.

Sandoval, I.O., Petsagkourakis, P., and del Rio-Chanona, E.A. (2021). Constrained Control with Neural Feedback Policies in DiffEqFlux. In *JuliaCon 2021*.

Schürer, R. (2003). A comparison between (quasi-)Monte Carlo and cubature rule based methods for solving high-dimensional integration problems. *Mathematics and Computers in Simulation*, 62(3-6), 509–517. doi:10.1016/S0378-4754(02)00250-1.