Accelerating Dynamical System Simulations with Contracting and Physics-Projected Neural-Newton Solvers

Samuel Chevalier[1]   SCHEV@ELEKTRO.DTU.DK
Jochen Stiasny[1]   JBEST@ELEKTRO.DTU.DK
Spyros Chatzivasileiadis[1]   SPCHATZ@ELEKTRO.DTU.DK
[1] *Technical University of Denmark, Kgs. Lyngby, Denmark*

Abstract

Recent advances in deep learning have allowed neural networks (NNs) to successfully replace traditional numerical solvers in many applications, thus enabling impressive computing gains. One such application is time domain simulation, which is indispensable for the design, analysis and operation of many engineering systems. Simulating dynamical systems with implicit Newton-based solvers is a computationally heavy task, as it requires the solution of a parameterized system of differential and algebraic equations at each time step. A variety of NN-based methodologies have been shown to successfully approximate the trajectories computed by numerical solvers at a fraction of the time. However, few previous works have used NNs to model the numerical solver itself. For the express purpose of accelerating time domain simulation speeds, this paper proposes and explores two complementary alternatives for modeling numerical solvers. First, we use a NN to mimic the linear transformation provided by the inverse Jacobian in a single Newton step. Using this procedure, we evaluate and project the exact, physics-based residual error onto the NN mapping, thus leaving physics "in the loop". The resulting tool, termed the Physics-pRojected Neural-Newton Solver (PRoNNS), is able to achieve an extremely high degree of numerical accuracy at speeds which were observed to be up to 31% faster than a Newton-based solver. In the second approach, we model the Newton solver at the heart of an implicit Runge-Kutta integrator as a *contracting map* iteratively seeking a fixed point on a time domain trajectory. The associated recurrent NN simulation tool, termed the Contracting Neural-Newton Solver (CoNNS), is embedded with training constraints (via CVXPY Layers) which guarantee the mapping provided by the NN satisfies the Banach fixed-point theorem; successive passes through the NN are therefore guaranteed to converge to a unique, fixed point. Explicitly capturing the contracting nature of Newton iterations leads to significantly increased NN accuracy relative to a vanilla NN. We test and evaluate the merits of both PRoNNS and CoNNS on three dynamical test systems.

Keywords: Dynamical Simulation, Recurrent Neural Networks, Runge-Kutta, Contraction

1. Introduction

Across many applications, Neural Networks (NNs) are being increasingly constructed, and subsequently utilized, as iterative self-mapping functions (Chen et al., 2018):

$$x^{(i+1)} = f(x^{(i)}, \theta), \tag{1}$$

where the NN mapping $f : \mathbb{R}^n \to \mathbb{R}^n$ is parameterized by some input θ. Residual Networks (ResNets) (He et al., 2016), normalizing flows (Kobyzev et al., 2020), and even standard Recur-

rent Neural Networks (RNNs) (Yu et al., 2019) all approximately leverage this self-mapping structure(Chen et al., 2018). The observation that (1) can be interpreted as a forward Euler step (Lu et al., 2018; Haber and Ruthotto, 2017) of some underlying continuous dynamical system has lead to the re-emergence (Massaroli et al., 2020; Cohen and Grossberg, 1983) of research which treats NNs as dynamical systems (Li et al., 2019). Notable recent works include, for example, Neural ODEs (Chen et al., 2018) with its many variants (Massaroli et al., 2020; Tzen and Raginsky, 2019; Poli et al., 2019; Zhang et al., 2019), and Deep Equilibrium Models (Bai et al., 2019, 2020). These approaches, collectively referred to as Continuous Deep Learning Models (CDLMs), provide a variety of computational and modeling benefits, but they are also able to directly leverage a century worth of tools from the differential equations community.

The CDLM perspective arises by treating (1) as a discretized Euler step of some continuous system; in the limit, (1) transforms into a true derivative (i.e., $\dot{x} = f(x, \theta)$), and CDLMs naturally emerge (Chen et al., 2018). In this paper, we offer an alternative perspective: rather than having (1) (or its continuous counterpart) directly model the evolution of some system of interest, we instead use it to model the iterative steps taken by the numerical *solver* of the system. Learning the behaviour of a numerical solver (rather than the behaviour of a system itself) can be advantageous for a number of reasons, including the learned solver's ability to provide arbitrary levels of simulation accuracy. In this paper, we focus exclusively on implicit Runge-Kutta (IRK) integration methods which require the solution of a (potentially nonlinear) system of equations at each time step; such IRK methods are typically solved with a Newton-based root-finding tool and are therefore computationally expensive (Iserles, 2008). NN-based models can be used to help reduce this computational effort. Modeling Newton's method with a NN has two salient alternatives; one can use a NN Φ to either (i) predict the linear transformation provided by the inverted Jacobian inside of a Newton loop, or (ii) directly predict iterative state updates via $x^{(i+1)} \leftarrow \Phi(x^{(i)})$. In this paper, we outline novel frameworks for both of these approaches, and we compare and contrast their merits.

Predicting linear transformations: We first use a NN to predict the expensive linear transformation (i.e., Jacobian inverse J^{-1}) mappings which are at the computational heart of a Newton routine. In doing so, we exploit the fact that Newton steps can include some degree of error and still iteratively arrive at the desired root. In this paper, we use a NN to learn these linear transformations, and then we iteratively project the residual error of future time domain trajectory points onto these linear transformations. We thus refer to the resulting tool as the Physics-pRojected Neural-Newton Solver (PRoNNS), since it projects physics-based residuals directly into the NN at each iteration.

Directly predicting state updates: We also explore removing the physics-based residual function altogether to learn a fully data-driven iterative solver. By directly modeling a numerical solver, we can equip (1) with a particular mathematical property that few dynamical systems actually possess: contraction. A self-mapping system contracts if a distinct set of inputs are farther apart than the outputs to which they map (Pata, 2019). Thus, we use (1) to model a contracting numerical solver. If certain conditions are satisfied (e.g., conditions for the Newton-Kantorovich Theorem (Ortega and Rheinboldt, 2000)), then Newton's method is guaranteed to contract around the fixed point it is seeking. By ensuring that the Newton solver always iterates in a locally contracting region, we can train a NN which both mimics Newton's iterative contracting behavior and, furthermore, is guaranteed to converge. The resulting NN is termed the Contracting Neural-Newton Solver (CoNNS).

Paper Contributions: In this paper, we design two approaches for modeling IRK numerical solvers. Through PRoNNS, (i) we develop a model which learns the linear transformations required at each Newton solver step, and (ii) we overcome the problem of training on residuals whose norms

decay to 0 by implementing a normalization procedure which preserves the mappings of the underlying linear transformations. Through CoNNS, (iii) we derive conditions under which a NN of the form (1) is guaranteed to contract and thus satisfy the Banach fixed point theorem, and (iv) we pose these contraction conditions as simple, semidefinite programming constraints which can be implemented with open source tools (CVXPY Layers (Agrawal et al., 2019)) inside of a NN training routine. Finally, (v) we provide a comparison between these alternative numerical solver modeling approaches and showcase their capabilities on several dynamical test systems.

2. Related work

The direct modeling of dynamical systems using NNs has a rich history (González-García et al., 1998; Milano and Koumoutsakos, 2002; Kosmatopoulos et al., 1995; Tiumentsev and Egorchev, 2019); subsequent simulation of the resulting parameterized models, though, requires the use of classical numerical solvers. Recently, attention has shifted towards constructing models which are capable of *directly* predicting the time domain response of a dynamical system, thus bypassing the need for a numerical solver. In particular, the so-called Physics Informed Neural Network (PINN) (Raissi et al., 2019), which uses physics-based sensitivities to regularize the training loss function, has achieved great success in extrapolating the solutions of Partial Differential Equations (PDEs), both in continuous and discrete time applications. A variety of follow up works (Pang et al., 2019; Shin et al., 2020; Lu et al., 2021; Misyris et al., 2020) have quickly improved upon the method, adding additional training constraints and extending it into new domains. In particular, (Roehrl et al., 2020) formally extends PINN modeling to ODEs; ODEs are the primary focus of this paper.

Rather than directly parameterizing a NN with an input time variable t, which is often necessary with PINNs, an alternative approximation scheme uses a *flow map* interpretation of dynamical systems (Ying and Candes, 2006); in this case, a NN-based flow map function learns to directly advance a dynamical system state from $x(t)$ to $x(t + \Delta t)$. This is accomplished with a feedforward NN in (Pan and Duraisamy, 2018) and a ResNet in (Qin et al., 2019). The standard ResNet approach is improved upon in (Liu et al., 2020), where a hierarchy of ResNets is used in order overcomes the problem of numerical stiffness in dynamical systems. NNs which mimic flow maps, however, must implicitly model both the dynamics of a system and the application of a numerical integration tool (Luchtenburg et al., 2014) (e.g., implicit Runge-Kutta) all at once, and there is no way to control the accuracy of the prediction they provide (other than increased training).

Ultimately, both PINNs and flow maps directly learn the trajectories associated with dynamical systems. Such trajectories, however, are usually computed with implicit numerical solvers. An even more fundamental approach to learning dynamical trajectories is to therefore directly learn the *iterative steps* taken by the associated numerical solver. While this approach has not been applied to the problem of time domain simulation, to the authors' knowledge, it has been applied in limited cases to other engineering problems (Baker, 2020; Mathia and Saeks, 1995). However, system physics has never been incorporated inside the iterative solver loop (as we do with PRoNNS), and the stability of the NN-based solver has not been explicitly addressed (as we consider with CoNNS).

The convergence of our second solver, CoNNS, coincides with the convergence of its RNN model. There is extensive work which analyzes the stability of RNNs, beginning with (Simard et al., 1988). Rigorous energy function formulations are offered in (Cao and Wang, 2005; Mostafa et al., 2011; Hu and Wang, 2002; Yi, 2013). When deriving CoNNS, we target contraction, which is a stronger notion than stability, but has sparsely been addressed in the discrete RNN literature until

more recently. (Steck, 1992) was one of the first papers which derived sufficient conditions for the contraction of a single layer RNN with bounded activation functions. Subsequently, (Mandic et al., 2000) developed analytical conditions related to the slope of the sigmoid activation function and the size of the weighting parameters. Most recently, (Miller and Hardt, 2018) investigated the stability of RNNs from the perspective of so-called λ-contractive sets. These methods are improved upon and analyzed in (Revay and Manchester, 2020), where projected gradient descent is used to project the NN model into a contracting space. In both (Miller and Hardt, 2018) and (Revay and Manchester, 2020), however, contraction of the RNN is utilized only as a conservative proxy to achieve stability; in neither case is convergence to a unique fixed point an explicit goal for the underlying RNN.

3. Mathematical background

We denote $\varsigma_m(\cdot)$ as the maximum singular value operator, i.e., $\varsigma_m(A) = \max\{\sqrt{\lambda(AA^T)}\}$ (Horn and Johnson, 1990). Notably, the largest singular value provides the bound $\|Ax\|_2 \leq \varsigma_m(A)\|x\|_2$.

3.1. Implicit Runge-Kutta Integration

We consider a time-invariant system of nonlinear ODEs given as $\dot{x}(t) \triangleq \frac{d}{dt}x(t) = f(x(t))$. The goal of time domain simulation (Liu et al., 2020) is to integrate $x(t)$ forward in time via

$$x(t + \Delta t) = x(t) + \int_t^{t+\Delta t} f(x(\tau))d\tau. \tag{2}$$

Since the closed form solution of (2) is rarely available, $x(t)$ is typically integrated using a Runge-Kutta time-stepping approach. Depending on how the parameters of the integration scheme are chosen, Runge-Kutta methods can take the form of many popular integrators (e.g., backward Euler); these methods can be classified as either explicit or implicit. In this work, we exclusively consider implicit integration schemes, since they are commonly used in many physics-based applications; they are, however, computationally expensive to solve. In the Implicit Runge-Kutta (IRK) method (Iserles, 2008), the future state $x(t + \Delta t)$ is written as the sum

$$x(t + \Delta t) = x(t) + \Delta t \sum_{i=1}^s b_i k_i, \tag{3}$$

where s is the number of "stages" associated with the IRK method, and k_i is a vector of trajectory derivatives at various points $t + c_i \Delta t$ between t and $t + \Delta t$. These derivative terms are computed as

$$k_i = f\left(x(t) + \Delta t \sum_{j=1}^s \alpha_{i,j} k_j\right), \quad i \in \{1, \ldots, s\}. \tag{4}$$

The parameters $\alpha_{i,j}$, b_i, and c_i are given coefficients from the Butcher tableau. Notably, the Runge-Kutta step given by (3)-(4) represents an implicit, nonlinear system of equations. Typically, an iterative root finding tool, such as Newton's method, is used to drive this system of nonlinear equations to 0 at each time step. Without loss of generality, this paper will focus on the trapezoidal integration method (Iserles, 2008), since it is the primary workhorse behind many ODE solvers (Milano, 2010). With trapezoidal integration, the implicit system of equations associated with (4) is given by

$$0 = k_2 - f(x(t) + \tfrac{\Delta t}{2}k_1 + \tfrac{\Delta t}{2}k_2)) \triangleq r(k_2), \tag{5}$$

where $k_1 = f(x(t))$. System (5) is typically solved by linearizing $r(k_2) \approx r(k_2^{(0)}) + J(k_2^{(0)})\Delta k_2$, setting the expression equal to 0, and then defining the iterative self-map

$$k_2^{(i+1)} = k_2^{(i)} - J(k_2^{(i)})^{-1} r(k_2^{(i)}) \triangleq g(k_2^{(i)}). \tag{6}$$

The primary computational bottleneck associated with IRK integration comes from solving nonlinear system (5) by iterating on (6). For convenience, $k \triangleq k_2$ in the remainder of this paper.

3.2. Fixed points of contracting systems

Definition 1 (Contraction Mapping (Pata, 2019)) *A function $f : \mathbb{R}^n \to \mathbb{R}^n$ is said to be contracting, or a contraction mapping, if, for any $x, y \in \mathbb{R}^n$, there exists $0 \le \mu < 1$ such that*

$$\|f(x) - f(y)\|_2 \le \mu \|x - y\|_2. \tag{7}$$

Theorem 2 (Banach Fixed-Point Theorem (Pata, 2019)) *Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be contracting on the complete metric space of $\mathbb{R}^n$. Then, f has a unique fixed point x^*, such that $x^* = f(x^*)$. Moreover, for any $x \in \mathbb{R}^n$, the sequence $f \circ \cdots f \circ f(x)$ converges to x^*.*

4. The Physics-pRojected Neural-Newton Solver (PRoNNS)

Panel **(a)** of Fig. 1 depicts the computational bottleneck associated with Newton iterations which are at the heart of any IRK method. In order to alleviate this bottleneck, we use a ReLU-based NN to learn the linear transformation which maps a residual error vector function $r(k^{(i)})$ to a Newton step via $\Delta k = -J(k^{(i)})^{-1} r(k^{(i)})$. ReLU-based NNs naturally provide piecewise linear input-output mappings, so they are particularly well-suited for learning the linear transformation codified by $J(k^{(i)})^{-1}$. Thus, we replace this linear transformation with ReLU-based NN Φ_p which takes the residual error, the current state trajectory estimate $k^{(i)}$, and the current state value $x(t)$ as inputs:

$$k^{(i+1)} = k^{(i)} - \Phi_p\left(r(k^{(i)}), k^{(i)}, x(t)\right). \tag{8}$$

Learning the linear mapping $J(k^{(i)})^{-1} r(k^{(i)})$ from simulated datasets is practically challenging, since the norm of the residual error vector r decays to 0 as k converges to its fixed point. To overcome this challenge, we normalize $r(k^{(i)})$ by its own norm, and we additionally scale the output step by the same size. The updated Newton iterations are equivalent to

$$k^{(i+1)} = k^{(i)} - \|r(k^{(i)})\| J(k^{(i)})^{-1} \frac{r(k^{(i)})}{\|r(k^{(i)})\|}. \tag{9}$$

When training a NN to mimic (9), we treat $r(k^{(i)})/\|r(k^{(i)})\|$ as an input, and the output emulates $J(k^{(i)})^{-1} r(k^{(i)})/\|r(k^{(i)})\|$. We re-scale the NN output by $\|r(k^{(i)})\|$ when implementing PRoNNS:

$$k^{(i+1)} = k^{(i)} - \|r(k^{(i)})\| \Phi_p\left(r(k^{(i)})/\|r(k^{(i)})\|, k^{(i)}, x(t)\right). \tag{10}$$

Despite the input scaling in (10), Φ_p is still trained to match the same linear transformation provided by the Jacobian transformation in (6). Finally, we note that even in the presence of linear transformation prediction error, routine (10) still has the potential to find the fixed point k^*. In the left-hand panel of Fig. 2, we show representative steps taken by a Newton solver. In the right-hand panel, we show how steps taken by PRoNNS, even in the presence of gross linearization error, can still converge to the fixed point of the residual function $r(k)$. Such convergence, however, is not guaranteed. In the following section, we introduce a NN-based solver whose convergence is guaranteed.
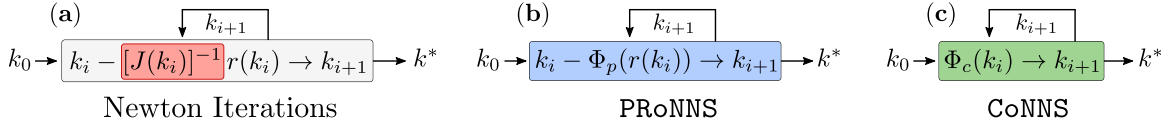
5

Figure 1: Panel (**a**) depicts classical Newton iterations associated with (6). Panel (**b**) depicts PRoNNS, where the inverted Jacobian has been replaced by a piecewise linear NN mapping. Panel (**c**) depicts CoNNS, where a NN mapping directly iterates on state updates.
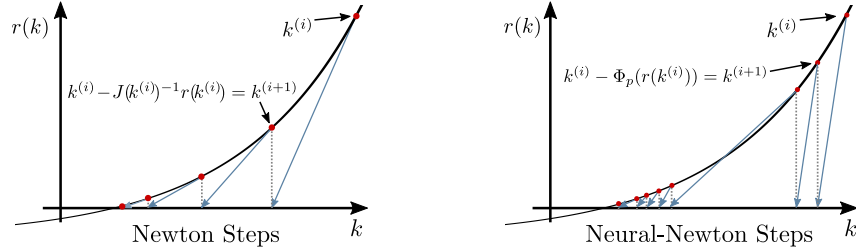

Figure 2: The left panel depicts typical Newton steps, while the right panel demonstrates how imperfect Neural-Newton steps, as taken by PRoNNS, can still potentially converge to the desired root.

5. The Contracting Neural-Newton Solver (**CoNNS**)

In this section, we build a NN which effectively emulates the Newton solver routine in (6), but we apply training constraints to ensure contraction of the NN mapping. We begin by casting the Newton iterations from (6) as a contracting system which is searching for a fixed point. Generally, Newton may not contract for an arbitrarily large time step Δt or arbitrarily chosen initial conditions. However, in this work, we assume these values are chosen such that Newton *does* represent a contracting system. Contraction in some region can be guaranteed by applying conditions from the Newton-Kantorovich Theorem (Ortega and Rheinboldt, 2000) or, more simply, by ensuring the gradient magnitude of the Newton map over some convex set is bounded to be less than 1. We may define analogous sufficient conditions in n-dimensions by taking the multivariable derivative of map (6) with respect to k_2 and ensuring the Jacobian singular values are bounded by unity:

$$\varsigma_m \left(J^{-1} \begin{bmatrix} (\frac{\partial}{\partial k_{2_1}} J) J^{-1} r & (\frac{\partial}{\partial k_{2_2}} J) J^{-1} r & \cdots & (\frac{\partial}{\partial k_{2_n}} J) J^{-1} r \end{bmatrix} \right) < 1, \ \forall k \in \mathcal{K}. \quad (11)$$

5.1. Conditions for guaranteed contraction of a neural network

We define an h-layer NN $k^{(i+1)} = \Phi_c(k^{(i)}, x(t))$, equipped with ReLU activation $\sigma(\cdot)$, via

$$k^{(i+1)} = \sigma \left(W_h \ldots \sigma \left(W_2 \sigma \left(W_1 k^{(i)} + U x(t) + b_1 \right) + b_2 \right) \ldots + b_h \right) \triangleq \Phi_c(k^{(i)}, x(t)). \quad (12)$$

We now present conditions which are sufficient to ensure the self-mapping routine (12) will contract.

Remark 3 *For all vectors $x, y \in \mathbb{R}^n$, the ReLU operator $\sigma : \mathbb{R}^n \to \mathbb{R}^n$ satisfies*

$$\|\sigma(x) - \sigma(y)\|_p \leq \|x - y\|_p, \ \forall p \in \{1, 2, ..., \infty\}. \quad (13)$$

Lemma 4 *Consider the ith layer of (12). If $\varsigma_m(W_i) < 1$, then this layer is a contraction mapping.*

6

Proof *We directly apply the definition of contraction (7) for arbitrary inputs x and y. The result yields $\|\sigma\left(W_i x + b_i\right) - \sigma\left(W_i y + b_i\right)\|_2 \leq \|W_i\left(x - y\right) + b_i - b_i\|_2 \leq \varsigma_m(W_i)\|x - y\|_2$.* ∎

Theorem 5 *A sufficient condition to ensure (12) satisfies the Banach Fixed-Point Theorem is*

$$\sup_{i \in \{1, \ldots, h\}} \varsigma_m(W_i) < 1. \tag{14}$$

Proof *Write (12) as a sequence of mappings, such that $g_1(k) = \sigma\left(W_1 k + U x(t) + b_1\right)$, $g_2(k) = \sigma\left(W_2 g_1(k) + b_2\right)$, ..., $g_h(k) = \sigma\left(W_h g_{h-1}(k) + b_h\right)$. By Lemma 4, each of these functions represents a contraction if (14) is satisfied. The composition of functions $\Phi \triangleq g_n \circ \cdots \circ g_2 \circ g_1$ which are individually contracting results in a new function which is also contracting. Thus, $\Phi(k, x(t))$ is contracting and necessarily satisfies the Banach FPT.* ∎

Directly constraining the singular values of a matrix (i.e., via (14)) is challenging. Instead, we seek to use semidefinite programming (SDP) tools which can efficiently constrain matrix eigenvalues. Furthermore, such tools are conveniently available through CVXPY Layers (Agrawal et al., 2019). Note that $\sigma(W) < 1$ implies $\lambda\left(W W^T\right) < 1$, and furthermore, $I - W I^{-1} W^T \succ 0$. By the Schur complement lemma (VanAntwerp and Braatz, 2000; Revay and Manchester, 2020)

$$\begin{bmatrix} I(1 - \epsilon) & W \\ W^T & I(1 - \epsilon) \end{bmatrix} \succeq 0 \implies \varsigma_m(W) < 1. \tag{15}$$

The contraction condition (14) can be satisfied for all square matrices by imposing (15) via conventional SDP tools; non-square matrices W_1 and W_h are dealt with by defining and dealing with augmented matrices, e.g., $\tilde{W}_1 = [W_1 \mid M_1] \in \mathbb{R}^{n \times n}$. At each training step, CVXPY Layers is used to optimally projects the unconstrained matrix W_i into a constrained space via

$$\min_{\hat{W}_i} \left\|W_i - \hat{W}_i\right\|_F^2, \ \forall i \in \{1, \ldots h\}, \quad \text{s.t. (15)}. \tag{16}$$

Constrained matrices $\hat{W}_i$ then replace their unconstrained counterparts W_i in (12). The projected training routine (16) guarantees (12) will converge to a unique fixed point k^*. Notably, SDP constraints are *only* enforced during training; NN evaluation uses standard matrix vector products and ReLU activations. Panel (**c**) of Fig. 1 portrays CoNNS, which iterates until convergence.

6. Numerical test results

In this section, we provide test results associated with three simulated systems: a two-state cubic oscillator (Liu et al., 2020), a three-state Hopf bifurcation (Liu et al., 2020), and a 10-state electrical power system, known as the Kundur system (Kundur et al., 1994).

Cubic Oscillator:
$$\dot{x} = -0.1x^3 + 2y^3$$
$$\dot{y} = -2x^3 - 0.1y^3$$

Hopf bifurcation:
$$\dot{\mu} = 0$$
$$\dot{x} = \mu x + y - x(x^2 + y^2)$$
$$\dot{y} = \mu y - x - y(x^2 + y^2)$$

Kundur system:
$$\dot{\delta}_i = \omega_i$$
$$\dot{\omega}_i = \hat{p}_i - \hat{d}_i \omega_i - \sum \hat{B}_{ij} \sin(\delta_i - \delta_j)$$
$$\forall i \in \mathcal{I}.$$

To collect training data, we defined a set of initial conditions associated with each system, and we randomly perturbed these conditions in order to generate strong system perturbations. We then simulated the resulting deterministic trajectories via the standard Newton-based trapezoidal integration method of (5)-(6). We used a step size Δt between 0.025s and 0.001s, and Newton convergence tolerance was set to $\epsilon = 10^{-9}$. Additional settings can be found directly in the code (Chevalier, 2021). After collecting 50 trajectories for each system, we appropriately trained both PRoNNS and CoNNS on the Newton step data. Training was performed with PyTorch (Paszke et al., 2019), and we used CVXPY Layers (Agrawal et al., 2019) to solve (16). We used Adam (Kingma and Ba, 2014) with learning rates set between 10^{-3} and 10^{-5} (see code). All simulation and training was performed on a Dell XPS laptop equipped with an Intel i7 CPU @ 2.60GHz 16 GB of RAM.

6.1. Results: **PRoNNS**

To model PRoNNS, we trained NNs containing three hidden layers; the NNs were respectively given 10 (cubic oscillator), 10 (Hopf), and 20 (Kundur) nodes per layer. We evaluated the performance of PRoNNS by testing it on 50 trajectories stemming from the same initial condition distributions as the training set. We then computed the 2-norm error across all variables and all time for every trajectory; the error associated with the jth trajectory, therefore, is $e_j = \|\hat{x}_i(t) - x_i(t)\|, \forall t, \forall i$, where $\hat{x}_i(t)$ is the ith state predicted by PRoNNS. Test results are depicted in Fig. 3. Notably, PRoNNS was able to accurately solve for trajectories across a wide region of initial conditions and in temporal regions well beyond its training (e.g., it was trained on 45 seconds of simulation data for the cubic/Hopf systems, but it could generalize well beyond these times). Error statistics are reported in Table 1; from these data, PRoNNS performed almost equally well on both unseen test trajectories and on training data. In Fig. 4, we depict the error accumulated by PRoNNS in various regions of state space for the cubic oscillator.
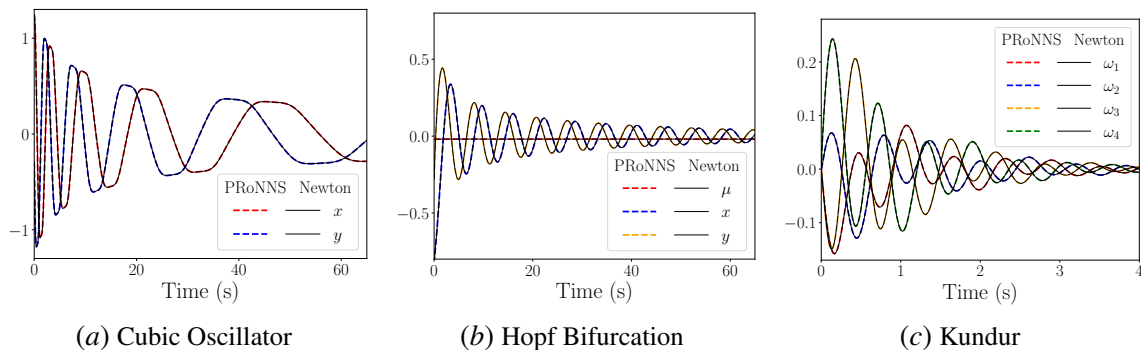

(a) Cubic Oscillator   (b) Hopf Bifurcation   (c) Kundur

Figure 3: Simulated trajectories pulled from test data, generated by Newton (benchmark) and PRoNNs.

Timing Analysis: In order to obtain a first impression of the computational properties, we conducted a timed comparison of the calculation of a single time step Δt using Newton iterations and PRoNNs for a range of states x_0, using thousands of repetitions and selecting the lowest run-time to reduce noise. For the cubic oscillator, PRoNNS required about 65% *more* time than the Newton iterations. For the Hopf Bifurcation system, the two methods matched each other very closely. Lastly, for the Kundur system, the largest among the test cases, PRoNNs required 31% *less* time than Newton; i.e., 0.075 ms for Newton's 0.108 ms. Absolute times are strongly influenced by Δt and x_0 as

Table 1: Training and test error, computed as the 2-norm of the trajectory approximation error.

Metric	Data	Cubic oscillator ($\times 10^{-3}$)		Hopf bifurcation ($\times 10^{-4}$)			Kundur ($\times 10^{-4}$)
		$\|x - \hat{x}\|_2$	$\|y - \hat{y}\|_2$	$\|\mu - \hat{\mu}\|_2$	$\|x - \hat{x}\|_2$	$\|y - \hat{y}\|_2$	$\|x - \hat{x}\|_2$
mean	Training	1.008	1.004	4.418×10^{-5}	3.422	3.406	1.397
mean	Test	0.978	0.969	4.456×10^{-5}	3.766	3.755	1.774
sd	Test	0.652	0.652	2.846×10^{-5}	3.188	3.155	1.055
max	Test	2.654	2.597	10.72×10^{-5}	9.083	9.199	4.376
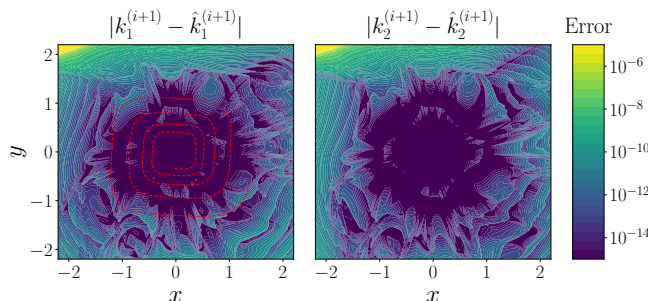
Figure 4: Error accumulated by PRoNNS trajectory steps in state space after training on the cubic oscillator. The red swirl shows the evolution of the *training* trajectory which evolved at the farthest region of state space.

well as the NN size, but the trend in the relative comparison matches the following consideration. Newton's method is dominated by system solve $J^{-1}r$ and scales with $\mathcal{O}(n^3)$. Meanwhile, evaluating h-layer NNs with m nodes per layer scales with $\sim \mathcal{O}(m^2)$, since $m \gg h$. Thus, computational benefit of PRoNNS is realized more saliently in large systems whose size satisfies $n > m^{2/3}$.

6.2. Results: CoNNS

We trained NNs containing four hidden layers with 40 (cubic), 50 (Hopf), and 100 (Kundur) nodes per layer. Notably, these NNs had to be several factors larger than the PRoNNS NNs in order to achieve acceptable error characteristics. We evaluated the performance of CoNNS by testing it on 50 trajectories stemming from the same distribution as the training set, and we bench-marked against an equivalently sized NN which was trained with no contraction constraints; both networks were trained to the same level of loss. Panels (a), (b), and (c) of Fig. (5) compare the performance of CoNNS to both the unconstrained NN and Newton for all three dynamical systems. CoNNS clearly outperforms the vanilla NN in the case of the cubic and Kundur systems. In panel (d), where a larger perturbation is applied to the system, the unconstrained NN shows sharp, incoherent trajectories which are very poor predictors of the true underlying dynamics.

Table 2 reports the cubic oscillator training and test results for both the constrained and unconstrained NNs.; similar statistics were computed for the Hopf and Kundur systems as well. Across all statistics, the results definitively suggest that CoNNS provides a more reliable prediction compared to the unconstrained NN predictions; furthermore, CoNNS required notably fewer iterations to converge. However, we required a relatively large NN to achieve these results (approximately 5x larger than PRoNNS in all cases), and the time domain error was relatively larger than the error accumulated by PRoNNS; compare, for instance, Fig. panel (c) of Fig 3 and panel (c) of Fig. 5.
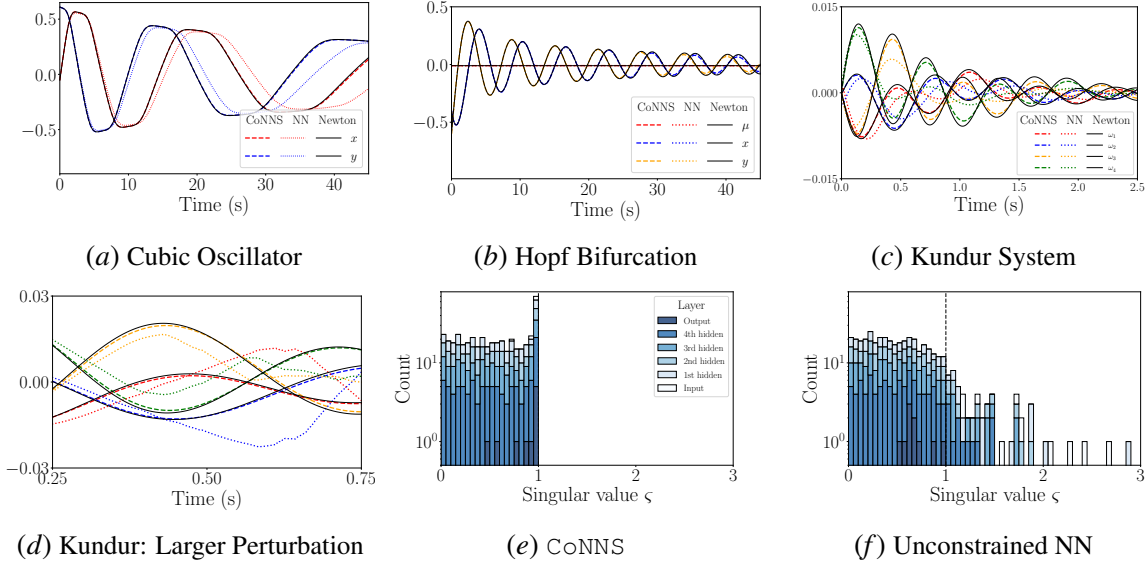
(*a*) Cubic Oscillator   (*b*) Hopf Bifurcation   (*c*) Kundur System

(*d*) Kundur: Larger Perturbation   (*e*) CoNNS   (*f*) Unconstrained NN

Figure 5: Panels (a)-(d) show system trajectories simulated with Newton, CoNNS and an unconstrained NN. Singular values of the Kundur NN weighting matrices are shown in panel (e)-(f).

Table 2: Training and test trajectory error. Iterations is the cumulative count of passes through CoNNS or the unconstrained NN for a given trajectory (Newton required a maximum of 8950 iterations, for reference).

Metric	Data	Cubic oscillator (*constrained*)			Cubic oscillator (*unconstrained*)		
		$\|x - \hat{x}\|_2$	$\|y - \hat{y}\|_2$	iterations	$\|x - \hat{x}\|_2$	$\|y - \hat{y}\|_2$	iterations
mean	Training	0.3685	0.3620	12873	7.8412	7.9659	216531
mean	Test	0.7297	0.7279	13177	8.7604	8.8905	213619
sd	Test	0.8200	0.8230	1728	5.4595	5.2871	21309
max	Test	4.0938	4.2462	16413	21.9004	22.5338	237974

7. Conclusion

With the goal of accelerating time domain simulation speeds, this paper developed two learning-based methods for emulating numerical solvers. The Physics-pRojected Neural-Newton Solver (PRoNNS) modeled inverted Jacobian transformations, and it was found to very successfully emulate trapezoidal integration, accumulating very low degrees of trajectory error ($\leq 10^{-3}$) across hundreds of tests, and achieving up to a 31% speed-up over a Newton-based benchmark. This result was achieved on a relatively small, 10-state system, and computational advantages will certainly scale with system size. While no convergence guarantees were derived, PRoNNS was found to reliably converge to meaningful solutions. We also developed a fully data-driven solver, termed the Contracting Neural-Newton Solver (CoNNS), which incorporated training constraints to guarantee iterative convergence. CoNNS was found to consistently outperform a vanilla-NN benchmark, offering an order of magnitude more accurate test performance on both the cubic oscillator and Kundur systems. This level of accuracy, however, did not meet the level of PRoNNS, and it came at the cost of incorporating $5\times$ more hidden neurons, resulting in slower-than-Newton performance.

References

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J. Zico Kolter. Differentiable convex optimization layers. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/9ce3c52fc54362e22053399d3181c638-Paper.pdf.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *arXiv preprint arXiv:1909.01377*, 2019.

Shaojie Bai, Vladlen Koltun, and J Zico Kolter. Multiscale deep equilibrium models. *arXiv preprint arXiv:2006.08656*, 2020.

Kyri Baker. Emulating ac opf solvers for obtaining sub-second feasible, near-optimal solutions. *arXiv preprint arXiv:2012.10031*, 2020.

Jinde Cao and Jun Wang. Global asymptotic and robust stability of recurrent neural networks with time delays. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 52(2):417–426, 2005. doi: 10.1109/TCSI.2004.841574.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*, 2018.

Samuel Chevalier. Implementation of Neural-Newton Solver for L4DC, 12 2021. URL https://gitlab.gbar.dtu.dk/schev/L4DC-Neural-Newton-Solvers.

Michael A. Cohen and Stephen Grossberg. Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):815–826, 1983. doi: 10.1109/TSMC.1983.6313075.

R. González-García, R. Rico-Martínez, and I.G. Kevrekidis. Identification of distributed parameter systems: A neural net based approach. *Computers & Chemical Engineering*, 22:S965–S968, 1998. ISSN 0098-1354. doi: https://doi.org/10.1016/S0098-1354(98)00191-4. URL https://www.sciencedirect.com/science/article/pii/S0098135498001914. European Symposium on Computer Aided Process Engineering-8.

Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

R.A. Horn and C.R. Johnson. *Matrix Analysis*. Cambridge University Press, 1990. ISBN 9780521386326.

Sanqing Hu and Jun Wang. Global stability of a class of discrete-time recurrent neural networks. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 49(8): 1104–1117, 2002. doi: 10.1109/TCSI.2002.801284.

Arieh Iserles. *A First Course in the Numerical Analysis of Differential Equations*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2 edition, 2008. doi: 10.1017/CBO9780511995569.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Ivan Kobyzev, Simon Prince, and Marcus Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

E.B. Kosmatopoulos, M.M. Polycarpou, M.A. Christodoulou, and P.A. Ioannou. High-order neural network structures for identification of dynamical systems. *IEEE Transactions on Neural Networks*, 6(2):422–431, 1995. doi: 10.1109/72.363477.

Prabha Kundur et al. *Power system stability and control*, volume 7. McGraw-Hill New York, 1994.

Qianxiao Li, Ting Lin, and Zuowei Shen. Deep learning via dynamical systems: An approximation perspective. *arXiv preprint arXiv:1912.10382*, 2019.

Yuying Liu, J Nathan Kutz, and Steven L Brunton. Hierarchical deep learning of multiscale differential equation time-steppers. *arXiv preprint arXiv:2008.09768*, 2020.

Lu Lu, Raphael Pestourie, Wenjie Yao, Zhicheng Wang, Francesc Verdugo, and Steven G Johnson. Physics-informed neural networks with hard constraints for inverse design. *arXiv preprint arXiv:2102.04626*, 2021.

Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *International Conference on Machine Learning*, pages 3276–3285. PMLR, 2018.

Dirk M. Luchtenburg, Steven L. Brunton, and Clarence W. Rowley. Long-time uncertainty propagation using generalized polynomial chaos and flow map composition. *Journal of Computational Physics*, 274:783–802, 2014. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp. 2014.06.029. URL https://www.sciencedirect.com/science/article/pii/ S0021999114004367.

D.P. Mandic, J.A. Chambers, and M.M. Bozic. On global asymptotic stability of fully connected recurrent neural networks. In *2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.00CH37100)*, volume 6, pages 3406–3409 vol.6, 2000. doi: 10.1109/ICASSP.2000.860132.

Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting neural odes. *arXiv preprint arXiv:2002.08071*, 2020.

K. Mathia and R. Saeks. Solving nonlinear equations using recurrent neural networks. In *World Congress on Neural Networks*, 1995.

Federico Milano. *Power system modelling and scripting*. Springer Science & Business Media, 2010.

Michele Milano and Petros Koumoutsakos. Neural network modeling for near wall turbulent flow. *Journal of Computational Physics*, 182(1):1–26, 2002. ISSN 0021-9991. doi: https://doi.org/10.1006/jcph.2002.7146. URL https://www.sciencedirect.com/science/article/pii/S0021999102971469.

John Miller and Moritz Hardt. Stable recurrent models. *arXiv preprint arXiv:1805.10369*, 2018.

George S. Misyris, Andreas Venzke, and Spyros Chatzivasileiadis. Physics-informed neural networks for power systems. In *2020 IEEE Power Energy Society General Meeting (PESGM)*, pages 1–5, 2020. doi: 10.1109/PESGM41954.2020.9282004.

Mohamad Mostafa, Werner G. Teich, and Jürgen Lindner. Global vs. local stability of recurrent neural networks as vector equalizer. In *2011 5th International Conference on Signal Processing and Communication Systems (ICSPCS)*, pages 1–5, 2011. doi: 10.1109/ICSPCS.2011.6140866.

J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Society for Industrial and Applied Mathematics, 2000. doi: 10.1137/1.9780898719468. URL https://epubs.siam.org/doi/abs/10.1137/1.9780898719468.

Shaowu Pan and Karthik Duraisamy. Long-time predictive modeling of nonlinear dynamical systems using neural networks. *Complexity*, 2018, 2018.

Guofei Pang, Lu Lu, and George Em Karniadakis. fpinns: Fractional physics-informed neural networks. *SIAM Journal on Scientific Computing*, 41(4):A2603–A2626, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Vittorino Pata. *Fixed point theorems and applications*, volume 116. Springer, 2019.

Michael Poli, Stefano Massaroli, Junyoung Park, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Graph neural ordinary differential equations. *arXiv preprint arXiv:1911.07532*, 2019.

Tong Qin, Kailiang Wu, and Dongbin Xiu. Data driven governing equations approximation using deep neural networks. *Journal of Computational Physics*, 395:620–635, 2019. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2019.06.042. URL https://www.sciencedirect.com/science/article/pii/S0021999119304504.

M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.10.045. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

Max Revay and Ian Manchester. Contracting implicit recurrent neural networks: Stable models with improved trainability. In Alexandre M. Bayen, Ali Jadbabaie, George Pappas, Pablo A. Parrilo, Benjamin Recht, Claire Tomlin, and Melanie Zeilinger, editors, *Proceedings of the 2nd Conference on Learning for Dynamics and Control*, volume 120 of *Proceedings of Machine Learning Research*, pages 393–403, The Cloud, 10–11 Jun 2020. PMLR. URL http://proceedings.mlr.press/v120/revay20a.html.

Manuel A Roehrl, Thomas A Runkler, Veronika Brandtstetter, Michel Tokic, and Stefan Obermayer. Modeling system dynamics with physics-informed neural networks based on lagrangian mechanics. *arXiv preprint arXiv:2005.14617*, 2020.

Yeonjong Shin, Jerome Darbon, and George Em Karniadakis. On the convergence and generalization of physics informed neural networks. *arXiv preprint arXiv:2004.01806*, 2020.

Patrice Y Simard, Mary B Ottaway, and Dana H Ballard. Fixed point analysis for recurrent networks. In *NIPS*, pages 149–159, 1988.

J.E. Steck. Convergence of recurrent networks as contraction mappings. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 3, pages 462–467 vol.3, 1992. doi: 10.1109/IJCNN.1992.227131.

Yury Tiumentsev and Mikhail Egorchev. *Neural Network Modeling and Identification of Dynamical Systems*. Academic Press, 2019.

Belinda Tzen and Maxim Raginsky. Neural stochastic differential equations: Deep latent gaussian models in the diffusion limit. *arXiv preprint arXiv:1905.09883*, 2019.

Jeremy G. VanAntwerp and Richard D. Braatz. A tutorial on linear and bilinear matrix inequalities. *Journal of Process Control*, 10(4):363–385, 2000. ISSN 0959-1524. doi: https://doi.org/10.1016/S0959-1524(99)00056-6. URL https://www.sciencedirect.com/science/article/pii/S0959152499000566.

Zhang Yi. *Convergence analysis of recurrent neural networks*, volume 13. Springer Science & Business Media, 2013.

Lexing Ying and Emmanuel J Candes. The phase flow method. *Journal of Computational Physics*, 220(1):184–215, 2006.

Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A review of recurrent neural networks: Lstm cells and network architectures. *Neural Computation*, 31(7):1235–1270, 2019. doi: 10.1162/neco_a_01199.

Tianjun Zhang, Zhewei Yao, Amir Gholami, Kurt Keutzer, Joseph Gonzalez, George Biros, and Michael Mahoney. Anodev2: A coupled neural ode evolution framework. *arXiv preprint arXiv:1906.04596*, 2019.