Taylor-Lagrange Neural Ordinary Differential Equations: Toward Fast Training and Evaluation of Neural ODEs

Franck Djeumou[*1] , **Cyrus Neary** [*1] , **Eric Goubault**[2] , **Sylvie Putot**[2] , **Ufuk Topcu**[1]
[1]The University of Texas at Austin, United States
[2]LIX, CNRS, École Polytechnique, Institut Polytechnique de Paris, France
{fdjeumou, cneary, utopcu}@utexas.edu, {goubault, putot}@lix.polytechnique.fr

Abstract

Neural ordinary differential equations (NODEs) – parametrizations of differential equations using neural networks – have shown tremendous promise in learning models of unknown continuous-time dynamical systems from data. However, every forward evaluation of a NODE requires numerical integration of the neural network used to capture the system dynamics, making their training prohibitively expensive. Existing works rely on off-the-shelf adaptive step-size numerical integration schemes, which often require an excessive number of evaluations of the underlying dynamics network to obtain sufficient accuracy for training. By contrast, we accelerate the evaluation and the training of NODEs by proposing a data-driven approach to their numerical integration. The proposed Taylor-Lagrange NODEs (TL-NODEs) use a fixed-order Taylor expansion for numerical integration, while also learning to estimate the expansion's approximation error. As a result, the proposed approach achieves the same accuracy as adaptive step-size schemes while employing only low-order Taylor expansions, thus greatly reducing the computational cost necessary to integrate the NODE. A suite of numerical experiments, including modeling dynamical systems, image classification, and density estimation, demonstrate that TL-NODEs can be trained more than an order of magnitude faster than state-of-the-art approaches, without any loss in performance.

1 Introduction

Neural ordinary differential equations (NODEs) have recently shown tremendous promise as a means to learn unknown continuous-time dynamical systems from trajectory data [Chen *et al.*, 2018]. By parametrizing differential equations as neural networks, as opposed to directly fitting the available trajectory data, NODEs provide compact representations of continuous-time systems that are memory-efficient and that are well understood; they allow the user to harness an existing wealth of knowledge from applied mathematics, physics, and engineering. For example, recent works have used NODEs as a means to incorporate physics-based knowledge into the learning of dynamical systems [Djeumou *et al.*, 2022a; Menda *et al.*, 2019; Gupta *et al.*, 2020; Cranmer *et al.*, 2020; Greydanus *et al.*, 2019; Finzi *et al.*, 2020; Zhong *et al.*, 2021]. Furthermore, NODEs have been used to define continuous normalizing flows – a class of invertible density models – to learn complex probability distributions over data [Chen *et al.*, 2018; Grathwohl *et al.*, 2019; Mathieu and Nickel, 2020; Salman *et al.*, 2018].

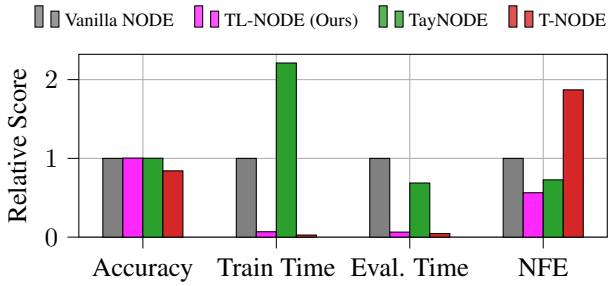

Figure 1: The proposed TL-NODE (magenta) achieves evaluation and training times that are more than an order of magnitude faster than state-of-the-art methods, without compromising any accuracy. The plot illustrates the results of using TL-NODE for a classification task on the MNIST dataset [Deng, 2012]. All scores are relative to those obtained by a vanilla NODE (grey), which uses an adaptive timestep numerical integrator. We compare against TayNODE [Kelly *et al.*, 2020] (green), and T-NODE (red) which uses a Taylor expansion for integration without the proposed correction employed by TL-NODE. The number of function evaluations (NFE) measures the regularity of the learned NODE (lower is better).

However, the training of NODEs can become prohibitively expensive [Grathwohl *et al.*, 2019; Kelly *et al.*, 2020; Finlay *et al.*, 2020]. In particular, every forward evaluation of the NODE requires the numerical integration of the underlying neural network parametrizing the system dynamics. Existing methods for the training of NODEs use off-the-shelf adaptive step-size numerical integration schemes for this purpose. However, in order to obtain sufficient accuracy, such integration schemes have been shown in practice to require an excessive number of evaluations of the underlying neural network. Furthermore, the severity of this problem has been

[*]These authors contributed equally.
The code is available at https://github.com/wuwushrek/TayLaNets.

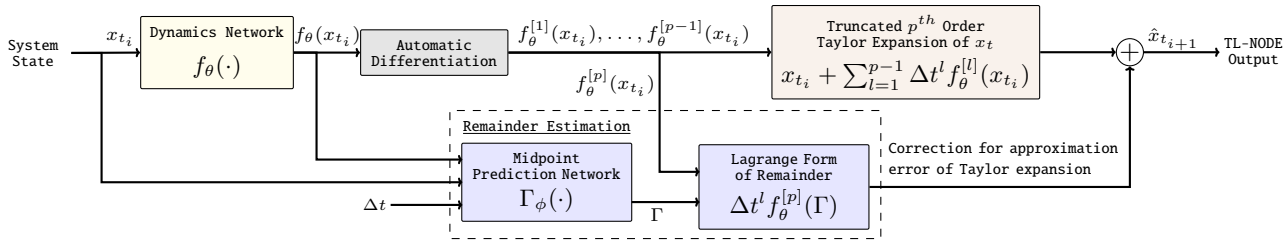

Figure 2: Taylor-Lagrange NODE (TL-NODE): an illustration of forward model evaluations. Given the system state x_{t_i} at time t_i, TL-NODE outputs a prediction of the state $\hat{x}_{t_{i+1}}$ at future time $t_{i+1} = t_i + \Delta t$. The dynamics network (yellow) parametrizes the differential equation being modeled. We use a truncated Taylor expansion (brown) of the state dynamics x_t to predict the future state. A separate midpoint prediction network (blue) is trained to estimate the remainder of the expansion, which is used as a correction for the model's prediction.

shown to grow as the training of the neural ODE progresses; while the neural ODE learns to fit the available data, it does not learn a representation of the dynamics that is easy to integrate [Finlay *et al.*, 2020]. These computational issues render the training of neural ODEs on large datasets intractable, and they also prevent neural ODEs from being deployed in applications requiring repeated fast online predictions; such as for model-predictive control of robotic systems.

To address the above issues, we present Taylor-Lagrange NODEs (TL-NODEs); we use a truncated Taylor expansion of the underlying neural network to predict the future system state, and we train a separate neural network to correct this prediction according to the Lagrange form of the expansion's remainder. By training the second corrector network, the approach significantly reduces the computational cost necessary for accurate numerical integration, while ensuring little-to-no-loss in the accuracy of the model. Figure 2 illustrates the major components of TL-NODE, which are discussed below.

(1) Taylor expansions for the numerical integration of the NODE. In order to integrate the NODE, we use a fixed-order Taylor expansion of the dynamical system in time. We take advantage of Taylor-mode automatic differentiation to efficiently compute the higher-order terms of the expansion on a GPU [Bettencourt *et al.*, 2019]. By specifying the number of terms to include in the expansion, we ensure that only a fixed number of evaluations of the underlying dynamics network are required per training step.

(2) Correcting the expansion's approximation error. We use the Lagrange form of the expansion's remainder to define a correction term, which greatly improves the accuracy of the predictions obtained from the truncated expansion. To estimate this approximation error efficiently, we propose to use a neural network to predict the so-called midpoint value – a point near the center of the expansion at which the approximation error can be evaluated exactly. While learning this midpoint value may, in general, be as complex as learning the neural ODE itself, we derive explicit formulas for the midpoint using assumptions on the regularity of the dynamics. These expressions reduce the complexity of the learning problem; only one unknown term in the expression must be learned. We provide upper bounds on the error of the proposed Taylor-Lagrange expansion, in terms of the error in the predicted midpoint value and the order of the expansion.

We demonstrate the effectiveness of the proposed approach through a suite of numerical experiments. The experimental tasks include the integration of known dynamics, learning to predict unknown dynamics, supervised classification, and density estimation. Figure 1 illustrates the result of applying TL-NODE to a classification task. Across all experiments we observe that the training and the evaluation of TL-NODEs is more than an order of magnitude faster than existing NODE methods, while incurring no loss in performance.

Related Work. Similarly to our work, a number of papers study how to reduce the computational cost of training neural ODEs. After [Chen *et al.*, 2018] initially presented the neural ODE, [Grathwohl *et al.*, 2019] proposed a stochastic estimator of the likelihood to reduce computational cost when using neural ODEs for continuous normalizing flows. [Kelly *et al.*, 2020; Finlay *et al.*, 2020; Pal *et al.*, 2021] propose additional regularization terms to learn neural ODEs that are easy to integrate. [Ghosh *et al.*, 2020] propose to regularize the neural ODE by randomly sampling the end time of the ODE during training. However, all of these works use off-the-shelf numerical integration algorithms for the forward evaluation of the NODE. By contrast, our work suggests a novel data-driven integration scheme, resulting in training times that are an order of magnitude faster than the current state-of-the-art.

Meanwhile, [Poli *et al.*, 2020] also suggest training an additional corrector neural network to speed up the numerical integration of NODEs. However, they do not present a technique that is able to apply such a corrector network during the training of the NODE. By contrast, we propose algorithms for the simultaneous training of the dynamics network and the remainder estimation network; this simultaneous training results not only in a speedup of the NODE evaluations, but also in a speedup of the NODE's training. Furthermore, we propose a method to simplify the learning of the remainder term by taking advantage of regularity assumptions on the system dynamics. This simplification leads to more efficient and generalizable learning of the correction term.

2 Background

We begin by introducing necessary background on ordinary differential equations (ODEs) and on neural ODEs.

Ordinary Differential Equations (ODEs). Let the function $f(x, t) : \mathbb{R}^n \times \mathbb{R}_+ \mapsto \mathbb{R}^n$ be *Lipschitz-continuous* in x

and t. An ordinary differential equation specifies the instantaneous change of a vector-valued signal of time $x : \mathbb{R}_+ \mapsto \mathbb{R}^n$.

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

We note that in general, the explicit dependence of $f(x, t)$ on t can be removed by adding a dimension to the state variable x. As such, throughout the remainder of the paper, we consider autonomous systems of the form $\dot{x}(t) = f(x(t))$. Furthermore, for notational simplicity we use the subscript notation x_t in place of $x(t)$.

Given some initial state $x_0 := x_{t_0} \in \mathbb{R}^n$ at initial time $t_0 \geq 0$, we wish to compute a solution to (1). In this work, we are specifically interested in predicting the value of the state x_t, at an arbitrary future point in time $T \geq t_0$. The value of x_T can be found by *integrating* or *solving* the ODE: $x_T = x_0 + \int_{t_0}^{T} f(x_s)\, ds$.

Neural Ordinary Differential Equations (NODEs). Neural ODEs (NODEs) are a class of deep learning models that use a neural network $f_\theta(x)$ to parametrize an ODE, and that subsequently use numerical integration algorithms to evaluate the model's outputs. More specifically, given a NODE input tuple (x_0, t_0, T) consisting of an initial state x_0, an initial time t_0, and a prediction time T, the output $\hat{x}_T := \mathrm{NODE}_\theta(x_0, t_0, T)$ of the NODE with parameter vector θ is given by $\mathrm{ODESolve}(f_\theta, x_0, t_0, T) \approx x_0 + \int_{t_0}^{T} f_\theta(x_s) ds$. Here, $\mathrm{ODESolve}(f_\theta, x_0, t_0, T)$ is a numerical approximation of the solution to the dynamics parametrized by $f_\theta(x)$.

We note that the particular algorithm used in place of $\mathrm{ODESolve}(\cdot)$ influences the model's accuracy, the computational cost of forward evaluations of the model, and the computational cost of training the model. Existing implementations of neural ODEs have typically relied on existing adaptive-step numerical integration algorithms for this purpose. By contrast, this work proposes the use of a novel Taylor expansion based method that uses data-drive estimations of the expansion's truncation error to evaluate and train neural ODEs efficiently and accurately.

3 Taylor-Lagrange Neural Ordinary Differential Equations (TL-NODE)

In this section we propose TL-NODEs for the efficient and accurate training and evaluation of neural ODEs.

3.1 Data-Driven Taylor-Lagrange Numerical Integration of the NODE Dynamics

Our objective is to efficiently and accurately evaluate neural ODEs through numerical integration of $f_\theta(\cdot)$. Toward this end, we propose to make direct use of the Taylor-Lagrange expansion of x_t in time. Taylor expansions are techniques used to approximate a function near a particular expansion point using polynomials. We use the term Taylor-Lagrange expansion to refer to the truncated Taylor expansion including the Lagrange form of the remainder. In order to obtain highly accurate expansions of x_t without requiring an excessive number of evaluations of $f_\theta(\cdot)$, we propose to train a separate neural network with parameters ϕ to estimate the remainder term in the expansion. We give a step-by-step description of the proposed methodology below.

Partitioning the prediction interval. We begin by partitioning the prediction interval $[t_0, T]$ into a collection of H sub-intervals $[t_i, t_{i+1}]$. For notational convenience, we assume the value of the integration time step is fixed, i.e. $\Delta t = t_{i+1} - t_i$ for all $i = 0, \ldots, H-1$. Given the value of the state x_{t_i} at time t_i, we compute an approximation of the value of the state at the next time step $\hat{x}_{t_{i+1}}$ via a p^{th} order Taylor-Lagrange expansion of x_t about time t_i, which we denote by $TL^{\Delta t}_{\theta,\phi}(x_{t_i})$. The output $\mathrm{TLNODE}_{\theta,\phi}(x_0, t_0, T)$ of the neural ODE is computed by iteratively using $TL^{\Delta t}_{\theta,\phi}(\cdot)$ to approximate the integral of $f_\theta(\cdot)$ over each sub-interval.

Expressing $TL^{\Delta t}_{\theta,\phi}(\cdot)$ in Terms of $f_\theta(\cdot)$. Note that because $f_\theta(x)$ estimates the time derivative of the system state x, the Taylor-Lagrange expansion of x_t may be expressed in terms of the *Taylor coefficients* $f^{[l]}_\theta(x)$, which are recursively defined through the equations $f^{[1]}_\theta(x) = f_\theta(x)$ and $f^{[l+1]}_\theta(x) = \frac{1}{l+1}[\frac{\partial f^{[l]}_\theta}{\partial x} f_\theta](x)$. Equation (2) accordingly presents the Taylor-Lagrange expansion of x_t about the point in time t_i, evaluated at the future time $t_{i+1} = t_i + \Delta t$.

$$\begin{aligned} TL^{\Delta t}_{\theta,\phi}(x_{t_i}) := x_{t_i} + \sum\nolimits_{l=1}^{p-1} \Delta t^l f^{[l]}_\theta(x_{t_i}) \\ + \mathcal{R}_\phi(f_\theta, x_{t_i}, \Delta t) \end{aligned} \tag{2}$$

The first two terms on the right hand side of (2) make up the *truncated* Taylor expansion of x_t, while $\mathcal{R}_\phi(f_\theta, x_{t_i}, \Delta t)$ denotes an estimation of the *remainder* of this truncated expansion. More specifically, $\mathcal{R}_\phi(f_\theta, x_{t_i}, \Delta t)$ estimates the approximation error of the p^{th} order expansion; if we could known this value exactly, (2) would provide an exact evaluation of the integral $x_{t_i} + \int_{t_i}^{t_{i+1}} f_\theta(x_s) ds$. Below we propose a methodology to learn to accurately estimate the value of this remainder term, given f_θ, x_{t_i}, and Δt as inputs.

Estimating the Remainder Term $\mathcal{R}_\phi(\cdot)$. To obtain accurate and generalizable estimations of the remainder term $\mathcal{R}_\phi(f_\theta, x_{t_i}, \Delta t)$, we begin by using Taylor's theorem to express it as $\mathcal{R}_\phi(f_\theta, x_{t_i}, \Delta t) = f^{[p]}_\theta(\Gamma)$. Here, $\Gamma \in \mathbb{R}^n$ denotes the *midpoint* of the Taylor-Lagrange expansion. More specifically, there exists some point in time ξ with $t_i \leq \xi \leq t_{i+1}$ such that when we define $\Gamma := x_\xi$, then $\Delta t^p f^{[p]}_\theta(\Gamma)$ provides the exact value of the approximation error of the expansion.

While no closed form expression for the midpoint Γ exists, we propose to learn to predict its value given the state x_{t_i} and the time step Δt. Learning to predict Γ directly as a function of these inputs is a challenging problem in general. We instead propose to use the result of Theorem 1, which provides a closed-form expression for Γ in terms of some unknown term $\bar{\Gamma} \in \mathbb{R}^n$. By taking advantage of this expression for Γ, we greatly simplify the task of learning to predict its value.

Theorem 1 (Simplified Midpoint Expression)**.** *If f_θ is a Lipschitz-continuous function, then there exists a function $\bar{\Gamma} : \mathbb{R}^n \times \mathbb{R}_+ \to \mathbb{R}^{n \times n}$ such that the midpoint value Γ of the Taylor-Lagrange expansion $TL^{\Delta t}_{\theta,\phi}(x_{t_i})$ is related to x_{t_i}, $f_\theta(x_{t_i})$, and $\bar{\Gamma}(x_{t_i}, \Delta t)$ through*

$$\Gamma = x_{t_i} + \bar{\Gamma}(x_{t_i}, \Delta t) \odot f_\theta(x_{t_i}), \tag{3}$$

where $\odot$ denotes matrix-vector multiplication.

Algorithm 1 Evaluating $\text{TLNODE}_{\theta,\phi}(x_0, t_0, T)$

Input: x_0, t_0, T
Parameter: θ, ϕ, p, H
Output: Model prediction $\hat{x}_T$.

1: $\hat{x}_{t_0} \leftarrow x_0$; $\Delta t \leftarrow \frac{T-t_0}{H}$
2: **for** $i = 0, 1, \ldots, H$ **do{** $t_i \leftarrow t_0 + i\Delta t$**}**
3: **for** $i = 0, 1, \ldots, H-1$ **do**
4: $\quad \Gamma \leftarrow \hat{x}_{t_i} + \bar{\Gamma}_\phi(\hat{x}_{t_i}, \Delta t) \odot f_\theta(\hat{x}_i)$
5: $\quad \hat{x}_{t_{i+1}} \leftarrow \hat{x}_{t_i} + \sum_{l=1}^{p-1} \Delta t^l f_\theta^{[l]}(\hat{x}_i) + \Delta t^p f_\theta^{[p]}(\Gamma)$
6: **end for**
7: **return** $\hat{x}_H$

Theorem 1 is obtained using tools from interval Taylor-Langrange based reachability analysis [Djeumou *et al.*, 2021]. More specifically, the equation given in (3) is derived from explicit formulas for the so-called *apriori* enclosure – a set derived from the local Lipschitzness of f_θ that is guaranteed to contain the value of $TL^{\Delta t}_{\theta,\phi}(x_{t_i})$. A proof of Theorem 1 is provided in the extended version of the paper [Djeumou *et al.*, 2022b]. We also prove that for linear dynamics, $\bar{\Gamma}$ does not depend on x_{t_i}.

Estimating the Midpoint Value. Given the result of Theorem 1, we propose to parameterize the unknown function $\bar{\Gamma}_\phi(\cdot)$ using a neural network with parameters ϕ. For notational simplicity, we use $\Gamma_\phi(x, \Delta t)$ to denote the value of the right hand side of (3) when $\bar{\Gamma}(\cdot)$ is approximated by $\bar{\Gamma}_\phi(\cdot)$. Given the predicted midpoint value $\Gamma_\phi(x, \Delta t)$, we estimate the remainder term of the p^{th} order Taylor-Lagrange expansion $TL^{\Delta t}_{\theta,\phi}(x_{t_i})$ as $\mathcal{R}_\phi(f_\theta, x_{t_i}, \Delta t) \approx \Delta t^p f_\theta^{[p]}(\Gamma_\phi(x_{t_i}, \Delta t))$.

The Proposed TL-NODE Evaluation Algorithm. Algorithm 1 summarizes the proposed approach for the numerical evaluation of neural ODEs. In lines 1 and 2, the prediction interval $[t_0, T]$ is broken into H sub-intervals. The for loop in lines $3-6$ iterates over these sub-intervals, and uses the midpoint prediction network $\Gamma_\phi(\cdot)$ to estimate the midpoint value (line 4), before using this estimate to approximate the state value $\hat{x}_{t_{i+1}}$ at the end of the sub-interval (line 5).

Bounding the Error of the TL-NODE Evaluation Algorithm. Given a fixed dynamics function $f_\theta(\cdot)$, we seek to bound the error on a p^{th} order Taylor-Lagrange expansion which uses a learned midpoint value predictor $\Gamma_\phi(\cdot)$ to estimate the expansion's remainder $\mathcal{R}_\phi(\cdot)$. Such an error bounds straightforwardly depends on how well $\Gamma_\phi(\cdot)$ approximates the true midpoint Γ, as described in Theorem 2. A proof of Theorem 2 is provided in the Appendix.

Theorem 2 (Integration Accuracy)**.** *If the learned midpoint predictor $\Gamma_\phi(\cdot)$ is a $\mathcal{O}(\eta)$ approximator to the midpoint Γ of the Taylor-Lagrange expansion of $TL^{\Delta t}_{\theta,\phi}(x_{t_i})$, then $\|x_{t_{i+1}} - TL^{\Delta t}_{\theta,\phi}(x_{t_i})\| \leq c\eta\Delta t^p$ for some $c > 0$ that depends on f_θ.*

A Note on the Evaluating the Taylor Coefficients. The Taylor coefficients $f_\theta^{[1]}(\cdot), \ldots, f_\theta^{[p]}(\cdot)$ can in principle be evaluated using repeated application of forward-mode automatic differentiation to iteratively compute the Jacobian-vector products $[\frac{\partial f_\theta^{[l]}}{\partial x} f_\theta](x)$. However, doing so would incur a time cost of $\mathcal{O}(exp(p))$. We instead use *Taylor mode* automatic differentiation, which computes the first p Taylor coefficients $f_\theta^{[1]}(\cdot), \ldots, f_\theta^{[p]}(\cdot)$ in a single pass, with a time cost of only $\mathcal{O}(p^2)$ or of $\mathcal{O}(p \log p)$, depending on the underlying operations involved [Griewank and Walther, 2008; Bettencourt *et al.*, 2019; Kelly *et al.*, 2020].

3.2 Training Taylor-Lagrange Neural Ordinary Differential Equations

Given a training dataset $\mathcal{D}$, we wish to train both components of the TL-NODE: the dynamics network $f_\theta(\cdot)$ and the midpoint prediction network $\Gamma_\phi(\cdot)$. To do so, we propose an algorithm that alternates between training each of the components via stochastic gradient descent while keeping the parameters of the other component fixed. We assume that each datapoint within the dataset $\mathcal{D} = \{(x_0^j, t_0^j, T^j, y^j)\}_{j=1}^{|\mathcal{D}|}$ is comprised of an initial state x^j, an initial time t_0^j, a prediction time T^j, and a labeled output value y^j.

Training the Dynamics Network $f_\theta(\cdot)$. We begin by holding the parameter vector ϕ of the midpoint prediction network to some fixed value $\hat{\phi}$, and training the dynamics network $f_\theta(\cdot)$ by solving the optimization problem (4) via stochastic gradient descent.

$$\min_\theta \sum_{(x_0^j, t_0^j, T^j, y^j) \in \mathcal{D}} \Big[\mathcal{L}(\text{TLNODE}_{\theta,\hat{\phi}}(x_0^j, t_0^j, T^j), y^j) + \lambda \sum_{i=0}^{H-1} ||\Delta t^p f_\theta^{[p]}(\Gamma_{\hat{\phi}}(\hat{x}_{t_i}, \Delta t))||^2 \Big] \tag{4}$$

Here, $\mathcal{L}(\cdot)$ is any differentiable loss function and $\hat{x}_{t_i}$ denotes integrator-estimated intermediate state at time t_i.

The summation in the second line of (4) measures the magnitude of the remainder terms $\mathcal{R}_\phi(\cdot)$ of the truncated Taylor-Lagrange expansions used for numerical integration. We may interpret this penalty term as having two purposes. Firstly, it acts as a regularizer that penalizes the higher-order derivatives of $f_\theta(\cdot)$ during training. Intuitively, by penalizing these higher order derivatives we encourage solutions that fit the data while also remaining as simple as possible. Secondly, the penalty term prevents the TL-NODE from using $\mathcal{R}_\phi(\cdot)$ to overfit the training data. By ensuring that the remainder term of the Taylor-Lagrange expansion remains small during training, we learn a dynamics function $f_\theta(\cdot)$ whose truncated expansions fit the training data as well as possible, while using $\mathcal{R}_\phi(\cdot)$ only for small corrections.

Training the Midpoint Prediction Network $\Gamma_\phi(\cdot)$. Recall that the midpoint prediction network $\Gamma_\phi(\cdot)$ plays a crucial role in accurately integrating the dynamics specified by $f_\theta(\cdot)$. So, as the parameters of the dynamics network $f_\theta(\cdot)$ are updated throughout training, our estimates of Γ, the midpoint of the Taylor-Lagrange expansion of $f_\theta(\cdot)$, should be updated accordingly. We thus propose to occasionally freeze the parameters of the dynamics network $\hat{\theta}$ in order to train $\Gamma_\phi(\cdot)$.

After fixing $\hat{\theta}$, we begin by generating a small dataset $\mathcal{D}_{\hat{\theta}}$. The datapoints of $\mathcal{D}_{\hat{\theta}}$ correspond to solutions of the

Algorithm 2 Training the TL-NODE

Input: Training dataset $\mathcal{D}$
Parameter: N_θ, N_ϕ, N_{train}, $N_{|\mathcal{D}_{\hat{\theta}}|}$
Output: Model parameters θ, ϕ

1: Initialize parameters θ, ϕ
2: **for** N_{train} steps **do**
3: Fix $\hat{\phi} \leftarrow \phi$
4: **for** N_θ steps **do**$\{$ $\theta \leftarrow \text{sgdStep}(\text{Eq. (4)}, \theta, \hat{\phi}, \mathcal{D})\}$
5: Fix $\hat{\theta} \leftarrow \theta$; $\{(x_0^j, t_0^j, T^j)\}_j \leftarrow \text{Sample}(\mathcal{D}, N_{|\mathcal{D}_{\hat{\theta}}|})$
6: $\mathcal{D}_{\hat{\theta}} \leftarrow \text{ODESolve}(f_{\hat{\theta}}, \{(x_0^j, t_0^j, T^j)\}_j)$
7: **for** N_ϕ steps **do**$\{$ $\phi \leftarrow \text{sgdStep}(\text{Eq. (5)}, \hat{\theta}, \phi, \mathcal{D}_{\hat{\theta}})\}$
8: **end for**
9: **return** θ, ϕ

ODE encoded by the fixed dynamics network $f_{\hat{\theta}}(\cdot)$. That is, for each $(x_0, t_0, T, y) \in \mathcal{D}_{\hat{\theta}}$ the output label y is given by $\text{ODESolve}(f_{\hat{\theta}}, x_0, t_0, T)$, where $\text{ODESolve}(\cdot)$ is a highly accurate adaptive-step ODE solver. Once the dataset $\mathcal{D}_{\hat{\theta}}$ has been generated, we train $\Gamma_\phi(\cdot)$ by using stochastic gradient descent to solve the optimization problem (5).

$$\min_{\phi} \sum_{(x_0^j, t_0^j, T^j, y^j) \in \mathcal{D}_{\hat{\theta}}} ||\text{TLNODE}_{\hat{\theta},\phi}(x_0^j, t_0^j, T^j) - y^j||^2 \quad (5)$$

The Proposed TL-NODE Training Algorithm. Algorithm 2 details the proposed training procedure. Throughout training we alternate between the following two subroutines: (lines 3-4) fix ϕ and take N_θ stochastic gradient descent steps to train the dynamics network $f_\theta(\cdot)$ according to (4), (lines 5-7) fix θ and take N_ϕ stochastic gradient descent steps to train the midpoint prediction network $\Gamma_\phi(\cdot)$ according to (5).

4 Numerical Experiments

We demonstrate the effectiveness of TL-NODE through several numerical experiments: the numerical integration of known dynamics, the learning of unknown dynamics, a supervised classification task, and a density estimation task. As an initial illustrative example we apply TL-NODE to linear dynamics. However, we note that the latter classification and density estimation tasks involve non-linear, time-dependent, and high-dimensional dynamics. Additional experimental details – including hyperparameter selection – are included in the extended version of the paper [Djeumou *et al.*, 2022b].

4.1 Modeling a Dynamical System

We begin by applying TL-NODE to the task of modeling a stiff dynamical system. More specifically, we use the proposed Taylor-Lagrange approach to learn, and to integrate, the ODE $\dot{x} = Ax$, where $x \in \mathbb{R}^2$ and $A \in \mathbb{R}^{2\times 2}$ has eigenvalues $\lambda_1 = -1$ and $\lambda_2 = -1000$.

Integration of Known Stiff Dynamics.
To examine the accuracy and robustness of the midpoint prediction network $\Gamma_\phi(\cdot)$, we begin by assuming the dynamics function $f(x) = Ax$ is known, and we use the proposed Taylor-Lagrange numerical integration method to predict future system states. We note that because we assume $f(\cdot)$ is known, there is no need to parameterize the system dynamics using a neural network $f_\theta(\cdot)$. However, we may still apply the method outlined in §3.2 to train $\Gamma_\phi(\cdot)$ to predict the approximation error of the Taylor-Lagrange expansions.

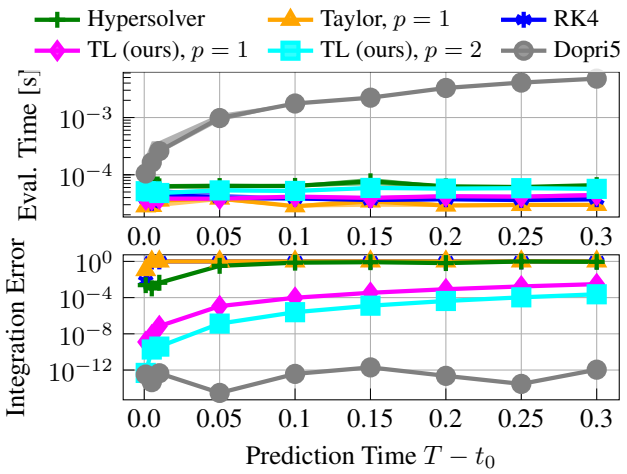

Figure 3: Numerical integration of known stiff dynamics. All algorithms other than Dopri5 use $H = 1$. Top: average time for numerical integration, as a function of the size of the prediction time interval. Bottom: average normalized integration error. Averages are taken with respect to 250 randomly sampled initial states.

Baselines. We apply both 1^{st} and 2^{nd} order Taylor expansions for numerical integration. For comparison, we include the results of a fixed-step RK4 method, an adaptive-step method (Dopri5), and the *Hypersolver* method [Poli *et al.*, 2020]. The tolerance parameters $rtol$ and $atol$ of the adaptive-step Dopri5 integrator are both set to $1.4e^{-12}$. We also plot the result of using a Taylor expansion for integration, without including the learned approximation error term.

Results. Figure 3 illustrates the numerical integration results. For brevity, in the figure we refer to the proposed Taylor-Lagrange method for integration as TL. We observe that TL-NODE enjoys lower integration error than all of the baseline methods except for Dopri5. However, Dopri5 requires computation times that are more than an order of magnitude higher than that of our method. We additionally observe that while the Hypersolver method requires similar computation time to TL-NODE, the error of its numerical integration results are several orders of magnitude higher. Furthermore, we note that for any prediction time intervals $T - t_0$ larger than $0.05(s)$, the fixed-step RK4, Truncated Taylor expansion method, and Hypersolver method all have normalized prediction errors values of 1.0 (the highest possible value). By contrast, our TL-NODE approach achieves an average error value on the order of 10^{-4}, even when $T - t_0 = 0.3(s)$. This demonstrataes the robustness of the proposed approach to the size of the prediction interval.

Finally, we note that the integration error of the truncated Taylor expansion method (yellow) is several orders of magnitude larger than that of TLN. The only difference between these methods is TLN's inclusion of the proposed correction term that learns the approximation error, demonstrating the gain in accuracy that this learned correction term provides.

Method	Train Accuracy (%)	Test Accuracy (%)	Train Time (min)	Eval. Time (ms)	NFE
TL-NODE (ours)	99.96	**98.23**	**2.55**	**1.04**	**62**
Vanilla NODE	99.33	97.87	42.7	16	110.6
TayNODE	99.29	98.02	94.3	11	80.00
RNODE	98.72	97.74	10.2	2.05	98.0
SRNODE*	**100.0**	98.08	98.1	-	259.0
STEER*	**100.0**	97.94	103	-	265.0

Table 1: MNIST image classification results.

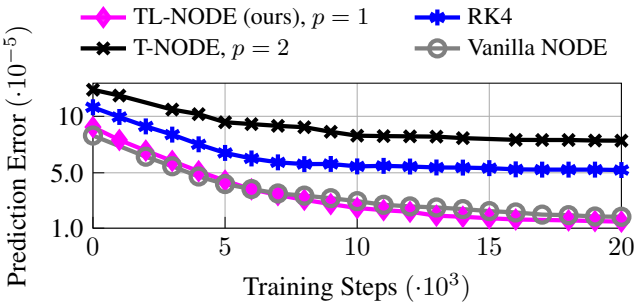

Figure 4: Predicting unknown dynamics over a prediction time step of $T - t_0 = 0.01(s)$. We plot the average mean square error of the predicted state as a function of the elapsed training steps.

Learning Unknown Dynamics

We now assume that the system dynamics $f(x) = Ax$ are unknown and train a TL-NODE to model the dynamical system.

Baselines. We compare to Vanilla NODE, which uses adaptive-step Dopri5 for numerical integration, to a NODE trained using fixed-step RK4, and to T-NODE – a version of our approach that also uses Taylor expansions for integration, but does not estimate their remainder term.

Results. Figure 4 illustrates the NODE's average prediction error as a function of the number of elapsed training steps. TL-NODE achieves smiliar prediction error values to the Vanilla NODE throughout training, while the prediction errors of the other two baseline methods are twice as large. The wall-clock training time for TL-NODE is 31.9s, for the Vanilla NODE it is 609.8s, for the RK4 NODE it is 35.8s, and for T-NODE it is 21.2s. Algorithm 2 effectively balances the training of TL-NODE's two components: $f_\theta(\cdot)$ and $\Gamma_\phi(\cdot)$. The result is a dynamics model that is as accurate as the Vanilla NODE trained using Dopri5, but whose training and evaluation times are much faster.

4.2 Supervised Classification

We train a TL-NODE model to perform classification on the MNIST dataset [Deng, 2012]. Our model follows the architecture of the neural ODE-based MNIST classifier presented in the work of [Kelly *et al.*, 2020] and further used in [Pal *et al.*, 2021] for benchmarking. Specifically, the model uses a two-layered neural network of size 100 and 728 (size of the images) with sigmoid-based non linearities to parameterize the dynamics function $f_\theta(\cdot)$. The NODE outputs propagate through a linear classifier to estimate of the image labels.

Baselines. We compare the proposed Taylor-Lagrange networks with state-of-the-art NODE algorithms. More specifically, we compare to RNODE [Finlay *et al.*, 2020] and TayNode [Kelly *et al.*, 2020] using the source code provided by the respective authors. We implicitly also compare our results with other regularization techniques such STEER [Ghosh *et al.*, 2020] and SRNODE [Pal *et al.*, 2021] thanks to the thorough experiments provided in [Pal *et al.*, 2021] for a similar model of the MNIST classification problem.

Results. Table 1 lists the experimental results. TL-NODE achieves evaluation and training times that are more than an order of magnitude faster than the baseline approaches, while also achieving the highest accuracy on the test dataset. TL-NODE also learns a dynamics network $f_\theta(\cdot)$ that requires the smallest number of function evaluations (NFE) when it is being numerically integrated using an adaptive-step integrator. The low NFE score of TL-NODE indicates that the regularization term in (4) is effective at producing learned dynamics networks $f_\theta(\cdot)$ that are easy to numerically integrate.

4.3 Density Estimation

We apply TL-NODEs to train continuous-normalizing-flow-based generative models [Chen *et al.*, 2018; Grathwohl *et al.*, 2019] to approximate the distribution of the MiniBooNE dataset [Roe *et al.*, 2005; Papamakarios *et al.*, 2017].

Method	Loss (nat)	Train Time (min)	NFE
TL-NODE (ours)	**9.62**	12.3	**167.9**
Vanilla NODE	9.74	59.7	183.8
TayNODE	9.75	148.3	168.2
RNODE	9.78	**10.32**	182.0

Table 2: Density estimation results.

Results. TL-NODE achieves the best loss score and the lowest required number of function evaluations (NFE) in comparison with the baseline approaches.

5 Conclusions

We present Taylor-Lagrange Neural Ordinary Differential Equations (TL-NODEs): a class of neural ODEs (NODEs) that use fixed-order Taylor expansions for numerical integration during NODE training and evaluation. TL-NODEs also train a separate neural network to predict the expansion's remainder, which is used as a correction term to improve the accuracy of the NODE's outputs. We demonstrate that TL-NODEs enjoy evaluation and training times that are an order of magnitude faster than the current state-of-the-art, without any loss in accuracy. Future work will aim to apply the accelerated NODE evaluation times to the online model-based control of unknown dynamical systems.

Acknowledgements

This work was supported in part by ARL W911NF2020132, AFOSR FA9550-19-1-0005, and NSF 1646522.

References

[Bettencourt *et al.*, 2019] J. Bettencourt, Matthew J. Johnson, and D. Duvenaud. Taylor-mode automatic differentiation for higher-order derivatives in jax. In *Workshop on Program Transformations for ML, NeurIPS*, 2019.

[Chen *et al.*, 2018] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 2018.

[Cranmer *et al.*, 2020] M. Cranmer, Sam Greydanus, Stephan Hoyer, Peter W. Battaglia, David N. Spergel, and Shirley Ho. Lagrangian neural networks. *ArXiv*, abs/2003.04630, 2020.

[Deng, 2012] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 2012.

[Djeumou *et al.*, 2021] Franck Djeumou, Abraham P Vinod, Eric Goubault, Sylvie Putot, and Ufuk Topcu. On-the-fly control of unknown smooth systems from limited data. In *2021 American Control Conference*. IEEE, 2021.

[Djeumou *et al.*, 2022a] Franck Djeumou, Cyrus Neary, Eric Goubault, Sylvie Putot, and Ufuk Topcu. Neural networks with physics-informed architectures and constraints for dynamical systems modeling. In *Learning for Dynamics and Control*. PMLR, 2022.

[Djeumou *et al.*, 2022b] Franck Djeumou, Cyrus Neary, Eric Goubault, Sylvie Putot, and Ufuk Topcu. Taylor-lagrange neural ordinary differential equations: Toward fast training and evaluation of neural odes. *ArXiv*, abs/2201.05715, 2022.

[Finlay *et al.*, 2020] Chris Finlay, J. Jacobsen, L. Nurbekyan, and Adam M. Oberman. How to train your neural ode: the world of jacobian and kinetic regularization. In *International Conference on Machine Learning*. PMLR, 2020.

[Finzi *et al.*, 2020] Marc Finzi, Ke Alexander Wang, and Andrew G Wilson. Simplifying hamiltonian and lagrangian neural networks via explicit constraints. *Advances in Neural Information Processing Systems*, 2020.

[Ghosh *et al.*, 2020] Arna Ghosh, Harkirat Singh Behl, Emilien Dupont, Philip H. S. Torr, and Vinay Namboodiri. Steer : Simple temporal regularization for neural odes. *ArXiv*, abs/2006.10711, 2020.

[Grathwohl *et al.*, 2019] Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Kristjanson Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *ArXiv*, abs/1810.01367, 2019.

[Greydanus *et al.*, 2019] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. *Advances in Neural Information Processing Systems*, 2019.

[Griewank and Walther, 2008] Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

[Gupta *et al.*, 2020] Jayesh K Gupta, Kunal Menda, Zachary Manchester, and Mykel Kochenderfer. Structured mechanical models for robot learning and control. In *Learning for Dynamics and Control*. PMLR, 2020.

[Kelly *et al.*, 2020] Jacob Kelly, Jesse Bettencourt, Matthew J Johnson, and David K Duvenaud. Learning differential equations that are easy to solve. *Advances in Neural Information Processing Systems*, 2020.

[Mathieu and Nickel, 2020] Emile Mathieu and Maximilian Nickel. Riemannian continuous normalizing flows. *ArXiv*, abs/2006.10605, 2020.

[Menda *et al.*, 2019] Kunal Menda, Jayesh K Gupta, Zachary Manchester, and Mykel J Kochenderfer. Structured mechanical models for efficient reinforcement learning. In *Workshop on Structure and Priors in Reinforcement Learning, International Conference on Learning Representations*, 2019.

[Pal *et al.*, 2021] Avik Pal, Yingbo Ma, Viral Shah, and Christopher V Rackauckas. Opening the blackbox: Accelerating neural differential equations by regularizing internal solver heuristics. In *International Conference on Machine Learning*. PMLR, 2021.

[Papamakarios *et al.*, 2017] George Papamakarios, Iain Murray, and Theo Pavlakou. Masked autoregressive flow for density estimation. *ArXiv*, abs/1705.07057, 2017.

[Poli *et al.*, 2020] Michael Poli, Stefano Massaroli, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Hypersolvers: Toward fast continuous-depth models. *ArXiv*, abs/2007.09601, 2020.

[Roe *et al.*, 2005] Byron P Roe, Hai-Jun Yang, Ji Zhu, Yong Liu, Ion Stancu, and Gordon McGregor. Boosted decision trees as an alternative to artificial neural networks for particle identification. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 2005.

[Salman *et al.*, 2018] Hadi Salman, Payman Yadollahpour, Tom Fletcher, and Nematollah Batmanghelich. Deep diffeomorphic normalizing flows. *ArXiv*, abs/1810.03256, 2018.

[Zhong *et al.*, 2021] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Benchmarking energy-conserving neural networks for learning dynamics from data. In *Learning for Dynamics and Control*. PMLR, 2021.