Compressing Deep ODE-Nets using Basis Function Expansions

Anonymous Author(s)
Affiliation
Address
email

Abstract

The recently-introduced class of ordinary differential equation network models (ODE-Nets) establishes a fruitful connection between deep learning and dynamical systems. In this work, we reconsider formulations of the weights as continuous-in-depth functions using linear combinations of basis functions. This perspective allows us to compress the weights through a change of basis, without retraining, while maintaining near state-of-the-art performance. In turn, both inference time and the memory footprint are reduced, enabling quick and rigorous adaptation between computational environments. Furthermore, our framework enables meaningful continuous-in-time batch normalization layers using function projections. The performance of basis function compression is demonstrated by applying continuous-in-depth models to (a) image classification tasks using convolutional units and (b) sentence-tagging tasks using transformer encoder units

1 Introduction

Neural network (NNs) models inspired by ordinary differential equations [6, 10, 39, 47, 51, 59, 60], often called also *ODE-Nets*, have recently emerged as a paradigm for modeling data. In this work, we introduce a methodology to compress ODE-Nets without retraining. To do so, we express the weights as continuous-in-depth functions using linear combinations of basis functions. Basis functions allow one to take advantage of powerful parameter transformations: (a) *adding parameters* can be used to grow the depth of ODE-Nets; (b) *decreasing parameters* can be used to compress model weights; (c) *function projections* allow to incorporate stateful layers (e.g.batch normalization) into ODE-Nets.

To be more concrete, let $\theta(t; \hat{\theta})$ be an arbitrary weight function that depends on time, or depth, t, which takes as argument the discrete learnable weights $\hat{\theta}$. We can represent this weight function as a linear combination of K (continuous-in-time) basis functions $\phi_k(t)$ as

$$\theta(t; \hat{\theta}) = \sum_{k=1}^{K} \phi_k(t)\, \hat{\theta}_k. \tag{1}$$

This representation allows us to perform in a systematic way various transformations from different parameter coefficients $\hat{\theta}^1$ to other coefficients $\hat{\theta}^2$ simply by transforming between bases.

In turn, basis transformations such as interpolation and projection allow us to compress models; i.e., we can systematically derive smaller networks from a single deep ODE-Net source model through a change of basis. Our approach is of interest since other techniques such as stochastic depth [27], layer dropout [14] and adaptive depth [12, 34], that are useful for discrete graph models, do not necessarily apply in a meaningful manner to ODE-Nets.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

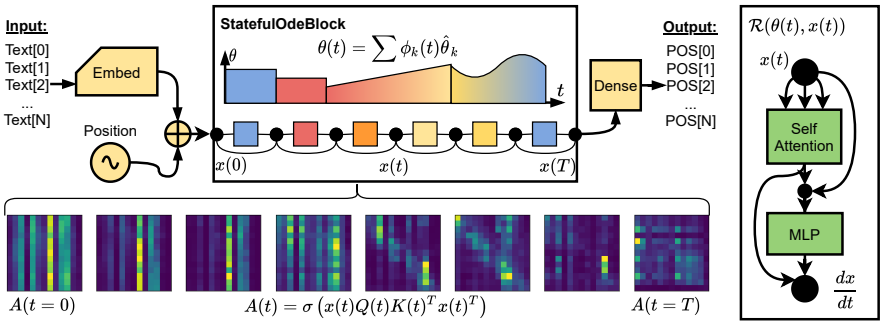


Figure 1: Sketch of a continuous-in-depth transformer-encoder. The model architecture consists of a sparse embedding layer followed by an OdeBlock that integrates the encoder to feed into a classification layer. The model graph is generated by nesting the residual $\mathcal{R}$ on the right into a time-integration scheme in the OdeBlock. The weights for each call to $\mathcal{R}$ are determined by evaluating the basis expansion. Internal hidden states evolve continuously during the forward pass. This is illustrated by a smoothly varying attention matrix which is extracted from a model trained in Sec. 6.3.

Further, this additional time-discretization on top of ODE-Nets elucidates an elegant way to incorporate stateful network layers, such as batch normalizations, into ODE-Nets to unlock state-of-the-art performance. We discretize both gradient parameters $\theta^g(t, \hat{\theta}^g)$ and state parameters $\theta^s(t, \hat{\theta}^s)$ using basis functions, where the gradient is denoted by g and the state by s. Given an update function $\mathcal{R}$, the continuous residual is functionally split into two components: the forward-pass output $\dot{x} = \mathcal{R}_x$ and the new state output $\theta^{s*} = \mathcal{R}_s$. The continuous-depth stateful ODE-Net satisfies the equations

$$\dot{x}(t) = \mathcal{R}_x\left(\theta^g(t), \theta^s(t), x(t)\right), \tag{2}$$

$$\theta^{s*}(t) = \mathcal{R}_s\left(\theta^g(t), \theta^s(t), x(t)\right), \tag{3}$$

subject to the initial condition $x_{in} = x(t=0)$. In order to discretize these equations, we solve $\dot{x}$ using methods for numerical integration (denoted by `scheme`), followed by a basis projection of $\hat{\theta}^{s*}$. Specifically, we yield the following input-output relation for the forward pass:

$$x_{out} = \texttt{StatefulOdeBlock}_x\left[\mathcal{R}_x, \texttt{scheme}, \Delta t, \phi\right]\left(\hat{\theta}^g, \hat{\theta}^s, x_{in}\right) \tag{4}$$

$$\hat{\theta}^{s*} = \texttt{StatefulOdeBlock}_s\left[\mathcal{R}_s, \texttt{scheme}, \Delta t, \phi\right]\left(\hat{\theta}^g, \hat{\theta}^s, x_{in}\right) \tag{5}$$

Given a loss function using $x_{out} = x(t=T)$, we jointly optimize for $\theta^g(t)$ and $\theta^s(t)$. The updates to $\theta^g(t)$ are computed by backpropagation of the loss through the OdeBlock with respect to its basis coefficients, $\partial x_{out}/\partial\hat{\theta}_g$. Optimizing $\hat{\theta}^s(t)$ involves projecting the update rule back onto the basis during every training step. In summary, this formulation provides a plug-and-play machinery that can be applied to any residual network $\mathcal{R}(\cdot)$, such as residual convolutional blocks with or without bottlenecks [25], wide variants [56], aggregated modules [53], or transformer encoders [52].

An example of a continuous-depth model using (4) is shown in Figure 1 for the part-of-speech (POD) tagging task considered in Section 6.3. Where a discrete transformer model would have a stacked chain of encoders, the continuous analogue has a single OdeBlock. The time-varying basis for parameters provides the depth-wise capacity of the model. Numerical integration of (2) generates a model graph with repeated subgraphs $\mathcal{R}$ whose weights are determined by evaluating $\theta(t; \hat{\theta}, \phi)$ at time t. The forward pass has a continuously varying hidden state that manifests, e.g., a smoothly varying attention matrix A(t) at different stages of the computation. We will show that with sufficient regularity, $\theta(t)$ can be effectively compressed while the model nearly maintains its accuracy.

The model formulated in (2) can be seen as an extension of the recently proposed ContinuousNet [47] and Galerkin ODE-Net [41]. In contrast to this prior work, we exploit properties of basis functions to compress ODE-Nets without retraining. Further, we are able to formulate continuous-time (CT) batch normalization layers. Indeed, stateful normalization layers are crucial for ODE-Nets to achieve state-of-the-art performance [20]. In the following, we refer to our model as ContinuousNetV2, and we show that, in addition to being able to compress our model without retraining, we are also able to achieve state-of-the-art performance on mainstream image classification problems.

Table 1: Comparison of our ContinuousNetV2 to various other dynamical system inspired models.

Model	Multi-level	Compression	Basis Function View	CT-BatchNorms
NODE [11]	✗	✗	✗	✗
Multi-level ResNet [5]	✓	✗	✗	✗
Galerkin ODE-Net [41]	✗	✗	✓	✗
ContinuousNet [47]	✓	✗	✓	✗
ContinuousNetV2 (ours)	✓	✓	✓	✓

Table 1 provides a high-level comparison of our results to other related dynamical-systems-inspired models (see Sec 2). To summarize our main contributions, we

- introduce a new methodology to compress ODE-Nets without retraining or revisting any data, based on parameter interpolation and projection (Sec. 3);
- propose a formulation for treating continuous-in-time stateful modules using function projections, which in turn enables us to introduce CT batch normalization layers for ODE-Nets (Sec. 4);
- introduce a theorem for the conditions required to learn continuous-in-depth weight functions in the discrete regime, and empirically demonstrate the theoretical result (Sec. 5);
- demonstrate the accuracy and compression performance for various image classification tasks using both shallow and deep ODE-Nets (Sec.6.2); and
- introduce a novel continuous-in-depth transformer and demonstrate the compression performance of this model on a natural language processing (NLP) task (Sec.6.3).

2 Related Work

The idea of "formulating models in continuous time and then discretize" [6, 22] has recently attracted attention both in the machine learning and physics-science community. Subsequently, numerous ODE and PDE-based network architectures [2, 8, 10, 18, 39, 47, 51, 59, 60], and continuous-time recurrent units [3, 13, 36, 48–50] have been proposed. Other research fronts motivated by this paradigm include normalizing flows [17, 29, 55], and stochastic differential equations [19, 28, 35, 38].

Exploiting the connections between ODE-Nets and the theory of dynamical systems and control is an active area of research [42, 45, 58] which has yielded more stable [7, 21, 22] and robust [1, 24, 37] models. In particular, this connection has motivated the development of more memory efficient training strategies [9, 15, 16, 46, 62] for ODE-Nets.

The idea of representing a continuous-in-time weight function as a linear combination of basis functions has recently been proposed by [42] and [47]. There, it has been noted that piecewise constant basis functions algebraically resemble residual networks and stacked ODE-Nets. Further, [42] used an orthogonal basis set to formulate a Galërkin Neural ODE. In contrast to their work, we use non-orthogonal basis sets and take advantage of basis transformations to compress ODE-Nets.

3 ODE-Net Compression via Basis Transformations

In this section, we discuss how basis transformations can be used to compress ODE-Nets. Basis transformations of continuous functions are a natural choice for compressing ODE-Nets because they marry well with continuous-time operations. The basis sets which we discuss have two parameters that specify the family of functions $\phi(t)$ and the number of functions K to be used. Hence, a basis is described as (ϕ, K). Here, we consider piecewise constant and piecewise linear basis functions.

- **Piecewise constant basis.** This basis consist of simple basis functions that assume a constant value for each of the elements of width $\Delta t = T/K$, where T is the 'time' at the end of the ODE-Net. The summation in Eq. (1) involves piecewise-constant "indicator" functions $\phi_k(t)$ satisfying

$$\phi_k(t) = \begin{cases} 1, & t \in [(k-1)\Delta t,\, k\Delta t) \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

- **Piecewise linear basis functions.** Next, we consider a basis with evenly spaced elements where each parameter $\hat{\theta}_k$ corresponds to the value of $\theta(t_k)$ at element boundaries $t_k = T(k-1)/(K-1)$. Each basis function is a "hat" function around the point t_k,

$$\phi_k(t) = \begin{cases} (t - k\Delta t)/\Delta t, & t \in [(k-1)\Delta t,\, k\Delta t] \\ 1 - (t - k\Delta t)/\Delta t, & t \in [k\Delta t,\, (k+1)\Delta t] \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

  Discontinuous piecewise linear functions are a closely related family of functions, wherein each control point has a left and right value. Both of these basis functions have compact and overlapping supports; i.e., they are not orthogonal.

Basis functions have a finite number of parameters, but enable us to generate compatible weights for any $t \in [0, T]$ needed by the integration of the forward pass. Given the coefficients $\hat{\theta}^1$ to a function $\theta^1(t)$ on a basis (ϕ^1, K_1), we can determine new coefficients $\hat{\theta}^2$ on a different space (ϕ^2, K_2). To illustrate this, Figure 2 shows different basis representations of a weight function. In practical terms, this transformation can change the actual floating-point values of parameters in memory, and changing K changes the total number of parameters, and hence storage requirement.

There are two types of transformation that we consider in this work: interpolation and projection. (Note that interpolation and projection do not necessarily give the same result for the same function.)

Interpolation. Some basis functions use control points for which parameters correspond to values at different t_k such that $\theta(t_k, \hat{\theta}) = \hat{\theta}_k$. Given θ^1, the parameter coefficients for θ^2 can thus be calculated by evaluating the θ^1 at the control points t_k^2:

$$\hat{\theta}_k^2 = \theta^1(t_k^2, \hat{\theta}^1) = \sum_{b=1}^{K_1} \phi_b^1(t_k^2)\hat{\theta}_b^1 \quad \text{for } k = 1, ..., K_2. \tag{8}$$

Interpolation only works with basis functions where the parameters correspond to control point locations. For piecewise constant basis functions, t_k corresponds to the cell centers, and for piecewise linear basis functions, t_k corresponds to the endpoints at element boundaries.

Projection. Function projection can be used with any basis expansion. Given the function $\theta^1(t)$, the coefficients $\hat{\theta}_k^2$ are solved by a minimization problem:

$$\min_{\hat{\theta}_k^2} \int_0^T \left(\theta^1(t, \hat{\theta}^1) - \theta^2(t, \hat{\theta}^2)\right)^2 \mathrm{d}t = \min_{\hat{\theta}_k^2} \int_0^T \left(\sum_{a=1}^{K_1} \hat{\theta}_a^1 \phi_a^1(t) - \sum_{k=1}^{K_2} \hat{\theta}_k^2 \phi_k^2(t)\right)^2 \mathrm{d}t. \tag{9}$$

The integral is evaluated using Gaussian quadrature over sub-cells. The SM includes the details of the numerical approximation of the loss and its solution. The ultimate calculation solves the same linear problem repeatedly for each tensor component of θ used by the call to $\mathcal{R}$.

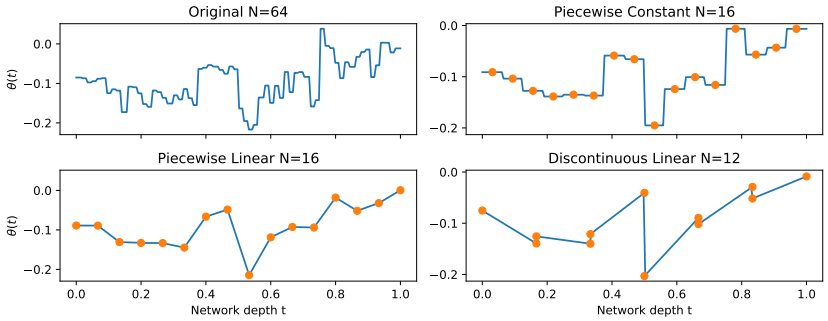


Figure 2: Real example of projecting a component of the query kernel from a continuous-depth transformer trained in Sec. 6.3. The model was trained with $K = 64$ piecewise constant basis functions (top left) and then projected to: $K = 16$ piecewise constant basis functions (top right), $K = 16$ piecewise linear basis functions (bottom left), and $K = 12$ discontinuous piecewise linear basis functions (bottom right). Circles denote the control points corresponding to parameters $\hat{\theta}_k$.

A Note on Complexity. Basis transformations operate directly on the parameter coefficients without any data inputs, model evaluations, or gradients. The total number of parameters for the two basis function models is $N_{\mathcal{R}} K_1$ and $N_{\mathcal{R}} K_2$, where $N_{\mathcal{R}}$ is the number of tensor components in one call to $\mathcal{R}$. Projection requires one matrix factorization and $N_{\mathcal{R}}$ applications of the factorization; the general-case complexity is $O\left((K_2)^3 + (K_2)^2 K_1 + N_{\mathcal{R}} K_1 K_2\right)$. Interpolation requires K_2 evaluations of ϕ^1 for each tensor component; the general-case complexity of interpolation is $O\left(N_{\mathcal{R}} K_2 K_1\right)$. K_1 and K_2 are proportional to the number of layers in a model and thus relatively small numbers, as compared to $N_{\mathcal{R}}$. The basis functions considered have compact support, which introduces sparsity that further reduces the complexities. Thus, because the network does not need to be evaluated on any data, basis transformations have very low time and memory cost.

4 ODE-Nets with Stateful Batch Normalization

For most tasks ODE-Nets require stateful modules such as BatchNorms to achieve top performance [20]. NNs are optimized by backpropagating on trainable parameters, however, stateful modules include specific update rules for parameters, $\theta^{s*} = \texttt{Update}(\theta^g, \theta^s, x_i)$. This complicates many aspects of implementing ODE-Nets, and contemporary work acknowledges that including BatchNorm layers in ODE-Nets is not obvious[54]. For instance, a recent approach emulated a piecewise-constant basis by multiplexing a list of stateful PyTorch BatchNorm objects in the integration loop[47].

Basis projection provides an elegant solution to formalize and implement continuous-in-depth stateful modules. Using a functional programming paradigm inspired by Flax [26], we replace the internal state update of the module with a secondary output. Let $\theta = \{\theta^g, \theta^s\}$, then the residual network can be split into two components, $\dot{x} = \mathcal{R}_x$, and $\theta^{s*} = \mathcal{R}_s$. For example, the continuous-depth BatchNorm function has two gradient-parameter functions, scale $s(t)$ and bias $b(t)$, and two state-parameter functions, mean $\mu(t)$ and variance $\sigma(t)$.

Algorithm 1: `StatefulOdeBlock` accumulates and projects state updates from the Runge-Kutta forward pass.

Data: Gradient and state parameters $\hat{\theta}^g$, $\hat{\theta}^s$,
Input x_{in}.
Initialize $\texttt{States} = \{\}$;
Let $x = x_{in}$;
for $t = 0$, $t < T$, $t \mathrel{+}= \Delta t$ **do**
  for $i \in$ *Runge-Kutta Stage* **do**
    Let $t_i = t + c_i \Delta t$;
    $x_i = \sum_j x + \Delta t a_{ij} k_j$;
    $k_i = \mathcal{R}_x(\theta^g(t_i, \hat{\theta}^g), \theta^s(t_i, \hat{\theta}^s), x_i)$;
    $\bar{\theta}_i^s = \mathcal{R}_s(\theta^g(t_i, \hat{\theta}^g), \theta^s(t_i, \hat{\theta}^s), x_i)$;
    $\texttt{States}.\text{append}(\{t_i, \bar{\theta}_s^i\})$;
  end
  Let $x = x + \Delta t \sum_i b_i k_i$;
end
$x_{out} = x$;
$\hat{\theta}^{s*} = \texttt{Project}(\texttt{States}, \phi, K)$;
Return: x_{out}, $\hat{\theta}^{s*}$;
Backward pass: Trace $\partial x_{out} / \partial \hat{\theta}^g$;

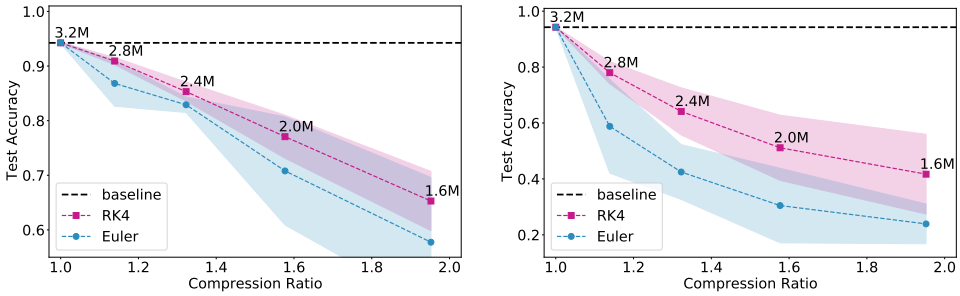
Eq.(3) is theoretically satisfied given a forward pass solution $x(t)$, but it requires an additional discretization to be implemented. Each call to $\mathcal{R}$ at time $t_i \in [0, T]$ during the forward pass outputs an updated state $\bar{\theta}_i^s = \mathcal{R}_s(t_i)$. This generates a point cloud $t_i, \bar{\theta}_i^s$, which is then projected back onto the basis set by minimizing the point-wise least-squared-error given by

$$\hat{\theta}^{s*} = \arg\min_{\hat{\theta}^{s*}} \sum_{i=1}^{N_{state}} \left(\bar{\theta}_i^s - \sum_{k=1}^{K} \phi_k(t_i) \hat{\theta}_k^{s*} \right)^2 . \tag{10}$$

Algorithm 1 describes the calculation of Equations 4 and 5, fused into one loop. For the forward Euler and piecewise constant case, this algorithm reduces to the update rule in a ResNet. This theoretical formalism and algorithm generalizes to any combination of stateful layer, basis set, and integration scheme. Interpolation and projection also apply to the coefficients $\hat{\theta}^s$.

5 Theoretical Bounds of Discretized Continuous Models

We briefly examine the advantages of certain integrators on compression from a theoretical point of view. For the sake of brevity, here we let $h = \Delta t$. A p-stage explicit Runge-Kutta method for an ODE $\dot{y}_\theta(t) = f(y(t), \theta(t))$ provides an approximation $y_{h,\theta}(t)$ of $y_\theta(t)$ for a given step size h through linear



(a) Compression using interpolation.

(b) Compression using projections.

Figure 3: Higher-order integrators introduce additional regularization into learned continuous weight functions. By experimenting on CIFAR-10, it is empirically observed that training with the RK4 scheme improves the compression-performance compared to using the forward Euler scheme.

interpolation between the generated points [23, Chapter II]: $y_{h,\theta}(t+h) = y_{h,\theta}(t) + h\sum_{i=1}^{p} b_i k_i$, where each k_i satisfies $k_i = f(y_{h,\theta}(t) + h\sum_{j=1}^{i-1} a_{ij}k_j, \theta(t+c_i h))$. In Theorem 1, we demonstrate that for any scalar loss function L, using a Runge-Kutta integrator *implicitly regularizes* towards smaller derivatives in f. Implicit regularization is of interest very generally [40], and in recent years it has received attention for neural network models [31, 43]. Since θ can be arbitrary, we consider finite differences in time. To this effect, recall that the m-th order forward differences are defined by $\Delta_h^m f(x,t) = \Delta_h^{m-1} f(x,t+h) - \Delta_h^{m-1} f(x,t)$, with $\Delta_h^0 f(x,t) = f(x,t)$.

Theorem 1 (Implicit regularization) *There exists a polynomial P depending only on the Runge-Kutta scheme satisfying $P(0) = 0$, and a smooth function $\bar{y}_\theta(t)$ depending on f, θ, and the scheme, such that for any $t \in [0,T]$, as $h \to 0^+$,*

$$L(y_{h,\theta}(t)) = L(\bar{y}_\theta(t)) + h^p \nabla L(\bar{y}_\theta(t)) \cdot e_{h,\theta}(t) + \mathcal{O}(h^{p+1}), \tag{11}$$

where $e'_{h,\theta}(t) = \frac{\partial f}{\partial y}(y_{h,\theta}(t), \theta(t)) e_{h,\theta}(t) + P \circ \mathcal{D}^p_{h,\theta}(t)$ and

$$\mathcal{D}^p_{h,\theta}(t) = \left\{ \left(\frac{\partial^l f_j}{\partial y^l} \otimes h^{-m} \Delta_h^m \right) f_j(y_{h,\theta}(t), \theta(t)) \right\}_{j=1,\ldots,d,\, l+m \le p+1}. \tag{12}$$

Theorem 1 demonstrates that Runge-Kutta integrators implicitly regularize toward smaller derivatives/finite differences of f of orders $k \le p+1$. To demonstrate the effect this has on compression, we consider the *sensitivity* of the solution y to $\theta(t)$, and hence to the choice of basis functions, using Gateaux derivatives. A smaller sensitivity to $\theta(t)$ allows for coarser approximations before encountering a significant reduction in accuracy. For any function $\varphi(t)$, let $D_\varphi y_{h,\theta}(t) = \frac{d}{d\epsilon} y_{h,\theta+\epsilon\varphi}(t)\big|_{\epsilon=0}$. A quick calculation (see Supplementary Material) reveals that

$$D_\varphi y_{h,\theta}(t) = \int_0^t e^{F_\theta(s,t)} \frac{\partial f}{\partial \theta}(y_{h,\theta}(s), \theta(s)) \varphi(s) \mathrm{d}s + \mathcal{O}(h^p),$$

where $F_\theta(s,t) = \int_s^t \frac{\partial f}{\partial y}(y_{h,\theta}(u), \theta(u)) \mathrm{d}u$. In light of Theorem 1, a reduced sensitivity can occur in one of two ways: (I) A higher-order integrator is used, causing higher-order derivatives to appear in the implicit regularization term $e_{h,\theta}$. By the Landau-Kolmogorov inequality [32], for any differential D and integer $m \ge 1$, $\|D^m f\| \ge c_m \|f\| (\|Df\|/\|f\|)^m$ for some $c_m > 0$. Hence, by implicitly regularizing towards smaller higher-order derivatives/finite differences, we should expect more dramatic reductions in the first-order derivative/finite difference as well. This effect is seen in Figure 3. (II) A larger step size h is used. Since doing so can lead to stability concerns during training, we later consider a refinement training scheme [5, 47] where h is slowly reduced.

6 Experiments

In this section, we present empirical results to demonstrate the accuracy and compression performance of ContinuousNetV2 for both image classification tasks and a NLP task. Each experiment was repeated with eight different seeds, and the figures report mean and variance. Additional details about the training process, resources used, and different model configurations are provided in the SM.

6.1 Compressing Shallow ODE-Nets for Image Classification Tasks

First, we consider shallow ODE-Nets, which have low parameter counts, in order to compare our models to meaningful baselines. We evaluate the performance on both MNIST [33] and CIFAR-10 [30]. Here, we train our ContinuousNetV2 using the classic Runge-Kutta (RK4) scheme and the multi-level refinement method proposed by [5, 47].

Results for MNIST. Due to the simplicity of this problem, we consider for this experiment an ODE-Net that has only a single OdeBlock with 8 units, and each unit has 12 channels. Our model has about $18K$ parameters and achieves about 99.4% test accuracy on average. Despite the lower parameter count, we outperform all other ODE-based networks on this task, as shown in Table 2.

Next, we compress our model by reducing the number of units down from 8 to 4, without retraining, fine-tuning or revisiting any data. The resulting model has approximately $10K$ parameters (a 45% reduction), while achieving about 99.2% test accuracy. We can compress the model even further to $7K$ parameters, if we are willing to accept a 2.6% drop in accuracy.

Results for CIFAR-10. Next, we demonstrate the compression performance on CIFAR-10. To establish a baseline that is comparable to other ODE-Nets, we consider a model that has two OdeBlocks with 8 units, and the units in the first block have 16 channels, while the units in the second block have 32 channels. Our model has about $207K$ parameters and achieves about 90.4% accuracy. Note, that our model has a similar test accuracy to that of a ResNet-20 with $270K$ parameters (a ResNet-20 yields about 91.25% accuracy [25]). In Table 3 we see that our model outperforms all other ODE-Nets on average, despite the comparable parameter count.

Again, we compress our model by reducing the number of units from 8 down to 4. The resulting model has $114K$ parameters and achieves about 89.9% accuracy. Despite the compression ratio of nearly 2, the performance of our compressed model is still better as compared to other ODE-Nets.

6.2 Compressing Deep ODE-Nets for Image Classification Tasks

Next, we demonstrate that we can also compress high-capacity deep ODE-Nets trained on CIFAR-10. We consider a model that has 3 ODE-blocks. The units within the three blocks have an increasing number of channels, [16,32,64]. (Additional results for CIFAR-100 are provided in the SM.)

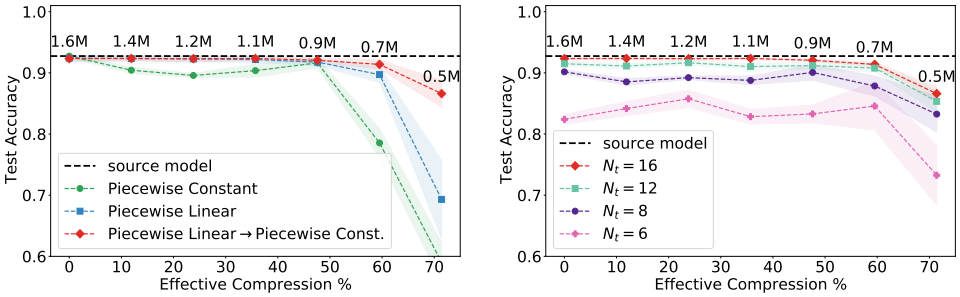
Table 4 shows results for CIFAR-10. Here we consider two configurations: (c1) is a model trained with refinement training, which has piecewise linear basis functions; (c2) is a model trained without refinement training, which has piecewise constant basis functions. It can be seen that model (c2)

Table 2: Compression performance and test accuracy of shallow ODE-Nets on MNIST.

Model	Best	Average	Min	# Parameters	Compression
NODE [11]	-	96.4%	-	84K	-
ANODE [11]	-	98.2%	-	84K	-
2nd-Order [41]	-	99.2%	-	20K	-
A4+NODE+NDDE [61]	-	98.5%	-	84K	-
ContinuousNetV2 (ours)	**99.6%**	**99.4%**	**99.3%**	18K	baseline
↪ (compressed)	99.4%	99.2%	99.1%	10K	45%
↪ (compressed)	97.8%	96.9%	95.7%	7K	61%

Table 3: Compression performance and test accuracy of shallow ODE-Nets on CIFAR-10.

Model	Best	Average	Min	# Parameters	Compression
HyperODENet [54]	-	87.9%	-	460K	-
SDE BNN (+ STL) [54]	-	89.1%	-	460K	-
Hamiltonian [51]	-	89.3%	-	264K	-
NODE [11]	-	53.7%	-	172K	-
ANODE [11]	-	60.6%	-	172K	-
A4+NODE+NDDE [61]	-	59.9%	-	107K	-
ContinuousNetV2 (ours)	**90.7%**	**90.4%**	**90.1%**	207K	baseline
↪ (compressed)	90.3%	89.9%	89.6%	114K	45%



(a) Compressing basis coefficients with fixed N_T.

(b) Decreasing N_T using the red line in (a).

Figure 4: In (a) we show the prediction accuracy on CIFAR-10 as a function of weight compression for models that are trained with different basis functions. It can be seen that there is advantage of using piecwise linear basis functions during training. In (b) we compare models with different numbers of time steps. Reducing the number of timesteps reduces the the number of FLOPs.

Table 4: Compression performance and test accuracy of deep ODE-Nets on CIFAR-10.

Model	Max	Average	Min	# Parameters	Compression
ResNet-110 [25]	-	93.4%	-	1.73M	-
ResNet-122-i [5]	-	93.8%	-	1.92M	-
MidPoint-62 [4]	-	92.8%	-	1.78M	-
ContinuousNet [47]	94.0%	93.8%	93.5%	3.19M	-
ContinuousNetV2 (c1)	93.0%	92.4%	92.1%	1.63M	baseline
↪ (compressed)	92.2%	91.8%	91.1%	0.85M	48%
ContinuousNetV2 (c2)	**94.4%**	**94.1%**	**93.8%**	1.63M	baseline
↪ (compressed)	69.9%	60.5%	52.7%	0.85M	48%

achieves high predictive accuracy, yet it yields a poor compression performance. In contrast, model (c1) is about 1.5% less accurate, but it shows to be highly compressible. We can compress the parameters by nearly a factor of 2, while increasing the test error only by 0.6%.

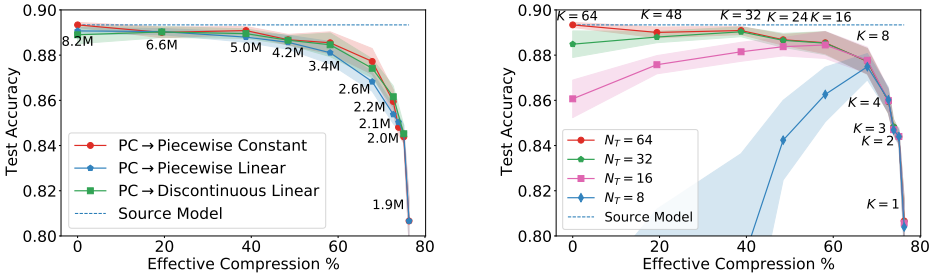
In Figure 4a, we show that the compression performance of model (c1) depends on the particular choice of basis set that is used during training and inference time. Using piecewise constant basis functions during training yields models that are slightly less compressible (green line), as compared to models that are using piecewise linear basis functions (blue line). Interestingly, the performance can be even further improved by projecting the piecewise linear basis functions onto piecewise constant basis functions during inference time (red line). In Figure 4b, we show that we can also decrease the number of time steps N_T during inference time, which in turn leads to a reduction of the number of FLOPs during inference time. For instance, reducing N_T from 16 to 8, reduces the inference time by about a factor of 2, while nearly maintaining the predictive accuracy. Recall, N_T refers to the number of time steps, which in turn determine the depth of the model graph.

6.3 Compressing Continuous Transformers

Finally, we consider a sequence-to-sequence tagging problem using a deep transformer-based encoder model. A discrete transformer-based encoder can be written as $x_{t+1} = x_t + T(q, x_t) + M(\rho, x_t + T(q, x_t))$ where $T(q, x)$ is self attention (SA) with parameters q (appearing twice due to the internal skip connection) and M is a multi-layer perceptron (MLP) with parameters ρ. Repeated transformer-based encoder blocks can be phrased as an ODE with the equation

$$\dot{x} = T(q(t), x) + M(\rho(t), x + T(q(t), x)). \tag{13}$$

Recognizing $\theta(t) = \{q(t), \rho(t)\}$, this formula can be directly plugged into the basis-functions and OdeBlocks to create a continuous-depth transformer, as illustrated in Fig. 1. Note that the OdeBlock is continuous along the *depth* of the computation; the sequence still uses discrete tokens and positional embeddings. With a forward Euler integrator, $\Delta t = 1$, a piecewise constant basis, and $K = 1$, Eq. 13 generates an algebraically equivalent graph to the discrete transformer.



(a) Compressing basis coefficients with fixed N_T.

(b) Decreasing N_T using the red line in (a).

Figure 5: In (a), we show the compression of a continuous-depth transformer for part-of-speech tagging. The transformer model can be compressed by a factor of 2, while nearly maintaining its predictive accuracy. Discounting the 1.8M embedding parameters, the smallest model achieves 98% compression, albeit losing accuracy. In (b) it is observed that the model graph can also be shortened.

We apply the encoder model to a part-of-speech tagging task, using the English-GUM treebank [57] from the Universal Dependencies treebanks [44]. Our model uses an embedding size of 128, with Key/Query/Value and MLP dimensions of 128 with a single attention head. The final OdeBlock has $K = 64$ piecewise constant basis functions and takes $N_T = 64$ steps.

In Figure 5a, we present the compression performance for different basis sets. Staying on piecewise constant basis functions yields the best performance during testing (red line). The performance slightly drops when we project the piecewise constant basis onto a piecewise linear basis (blue line). Projecting to a discontinuous linear basis (green line) performs approximately as well as the piecewise constant basis. In Figure 5b, we show the effect of decreasing the number of time steps N_T during inference time on the predictive accuracy. In the regime where $N_T \leq K$, there is no significant loss obtained by reducing K. Note that there is a divergence when $N_T > K$, where the integration method will skip over parameters. However, projection to $N_T \leq K$ incorporates information across a larger depth-window. Thus, projection improves graph shortening as compared to previous ODE-Nets.

7 Conclusion

In this work, we introduced a new ODE-based model that can be compressed without retraining, and thus quickly and seamlessly be adopted to different computational environments. This is achieved by formulating the weight function of our model as linear combinations of basis functions, which in turn allows us to take advantage of parameter transformations. In addition, this formulation also allows us to implement meaningful CT batch normalization layers that are crucial for achieving predictive accuracy comparable to ResNets. Indeed, our experiments showcase that our ContinuousNetv2 outperforms other recently proposed ODE-Nets, achieving 94.1% accuracy on CIFAR-10.

When using the (multi-level) refinement training scheme in combination with piecewise linear basis functions, we are able to compress nearly 50% of the parameters while sacrificing only a small amount of accuracy. On a natural language modeling task, we are able to compress nearly 98% of the parameters, while still achieving good predictive accuracy. Building upon the theoretical underpinnings of numerical analysis, we demonstrate that our compression method reliably generates consistent models without requiring retraining and *without needing to revisit any data.* Further, continuous-time formulations can also decrease the number of model steps, increasing inference speed in addition to decreasing the model size. However, a limitation of our approach is that the implicit regularization effect introduced by the (multi-level) refinement training scheme reduces the accuracy of the source model. Hence, future work should investigate improved training strategies as well as explore alternative basis function sets that improve the compression performance even further. We can also explore smarter strategies for compression and model graph shortening by pursuing hp-adaptivity algorithms. Further, it is interesting to explore the interplay of our methods with other compression techniques such as sparsification or quantification techniques.

Our work is foundational research and does not have any foreseeable negative societal impacts. Of positive impact, our techniques enable cheaply adapting models without requiring access to data.

References

[1] Srinivas Anumasa and PK Srijith. Improving robustness and uncertainty modelling in neural ordinary differential equations. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 4053–4061, 2021.

[2] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, pages 690–701, 2019.

[3] Bo Chang, Minmin Chen, Eldad Haber, and Ed H. Chi. AntisymmetricRNN: A dynamical system view on recurrent neural networks. In *Proceedings of the International Conference on Learning Representations*, 2019.

[4] Bo Chang, Lili Meng, Eldad Haber, Lars Ruthotto, David Begert, and Elliot Holtham. Reversible architectures for arbitrarily deep residual neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[5] Bo Chang, Lili Meng, Eldad Haber, Frederick Tung, and David Begert. Multi-level residual networks from dynamical systems view. In *Proceedings of the International Conference on Learning Representations*, 2018.

[6] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6571–6583, 2018.

[7] Marco Ciccone, Marco Gallieri, Jonathan Masci, Christian Osendorfer, and Faustino Gomez. Nais-net: Stable deep networks from non-autonomous differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[8] Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

[9] Talgat Daulbaev, Alexandr Katrutsa, Larisa Markeeva, Julia Gusak, Andrzej Cichocki, and Ivan Oseledets. Interpolation technique to speed up gradients propagation in neural odes. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 16689–16700. Curran Associates, Inc., 2020.

[10] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural ODEs. In *Advances in Neural Information Processing Systems*, pages 3140–3150, 2019.

[11] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes. 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.

[12] Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. In *International Conference on Learning Representations*, 2020.

[13] N Benjamin Erichson, Omri Azencot, Alejandro Queiruga, and Michael W Mahoney. Lipschitz recurrent neural networks. *arXiv preprint arXiv:2006.12070*, 2020.

[14] Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations*, 2020.

[15] Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam M Oberman. How to train your neural ODE. *arXiv preprint arXiv:2002.02798*, 2020.

[16] Amir Gholami, Kurt Keutzer, and George Biros. ANODE: unconditionally accurate memory-efficient gradients for neural ODEs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 730–736, 2019.

[17] Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. FFJORD: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

[18] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. In *Advances in Neural Information Processing Systems*, pages 15379–15389, 2019.

[19] Batuhan Güler, Alexis Laignelet, and Panos Parpas. Towards robust and stable deep learning algorithms for forward backward stochastic differential equations. *arXiv preprint arXiv:1910.11623*, 2019.

[20] Julia Gusak, Larisa Markeeva, Talgat Daulbaev, Alexander Katrutsa, Andrzej Cichocki, and Ivan Oseledets. Towards understanding normalization in neural {ode}s. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

[21] Eldad Haber, Keegan Lensink, Eran Treister, and Lars Ruthotto. IMEXnet a forward stable deep neural network. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 2525–2534, 2019.

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

[23] Ernst Hairer, Syvert P Nørsett, and Gerhard Wanner. *Solving ordinary differential equations. 1, Nonstiff problems*. Springer-Vlg, 1993.

[24] Yan Hanshu, Du Jiawei, Tan Vincent, and Feng Jiashi. On robustness of neural ordinary differential equations. In *Proceedings of the International Conference on Learning Representations*, 2019.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[26] Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2020.

[27] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pages 646–661. Springer, 2016.

[28] Junteng Jia and Austin R Benson. Neural jump stochastic differential equations. In *Advances in Neural Information Processing Systems*, pages 9847–9858, 2019.

[29] Jonas Köhler, Leon Klein, and Frank Noé. Equivariant flows: Sampling configurations for multi-body systems with symmetric energies. *arXiv preprint arXiv:1910.00753*, 2019.

[30] Alex Krizhevsky et al. Learning multiple layers of features from tiny images. 2009.

[31] J. Kukacka, V. Golkov, and D. Cremers. Regularization for deep learning: A taxonomy. Technical Report Preprint: arXiv:1710.10686, 2017.

[32] Man K Kwong and Anton Zettl. *Norm inequalities for derivatives and differences*. Springer, 2006.

[33] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[34] Xian Li, Asa Cooper Stickland, Yuqing Tang, and Xiang Kong. Deep transformers with latent depth. *Advances in Neural Information Processing Systems*, 33, 2020.

[35] Xuechen Li, Ting-Kam Leonard Wong, Ricky TQ Chen, and David Duvenaud. Scalable gradients for stochastic differential equations. *arXiv preprint arXiv:2001.01328*, 2020.

[36] Soon Hoe Lim. Understanding recurrent neural networks using nonequilibrium response theory. *arXiv preprint arXiv:2006.11052*, 2020.

[37] Soon Hoe Lim, N Benjamin Erichson, Liam Hodgkinson, and Michael W Mahoney. Noisy recurrent neural networks. *arXiv preprint arXiv:2102.04877*, 2021.

[38] Xuanqing Liu, Tesi Xiao, Si Si, Qin Cao, Sanjiv Kumar, and Cho-Jui Hsieh. Neural SDE: Stabilizing neural ode networks with stochastic noise. *arXiv preprint arXiv:1906.02355*, 2019.

[39] 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.

[40] M. W. Mahoney. Approximate computation and implicit regularization for very large-scale data analysis. In *Proceedings of the 31st ACM Symposium on Principles of Database Systems*, pages 143–154, 2012.

[41] Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting neural odes. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 3952–3963. Curran Associates, Inc., 2020.

[42] Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asma. Dissecting neural odes. In *34th Conference on Neural Information Processing Systems, NeurIPS 2020*. The Neural Information Processing Systems, 2020.

[43] B. Neyshabur. Implicit regularization in deep learning. Technical Report Preprint: arXiv:1709.01953, 2017.

[44] Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Jan Hajič, Christopher D Manning, Sampo Pyysalo, Sebastian Schuster, Francis Tyers, and Daniel Zeman. Universal dependencies v2: An evergrowing multilingual treebank collection. *arXiv preprint arXiv:2004.10643*, 2020.

[45] Katharina Ott, Prateek Katiyar, Philipp Hennig, and Michael Tiemann. Resnet after all: Neural {ode}s and their numerical solution. In *International Conference on Learning Representations*, 2021.

[46] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in Pytorch. 2017.

[47] Alejandro F Queiruga, N Benjamin Erichson, Dane Taylor, and Michael W Mahoney. Continuous-in-depth neural networks. *arXiv preprint arXiv:2008.02389*, 2020.

[48] Yulia Rubanova, Ricky T. Q. Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. In *Advances in Neural Information Processing Systems*, pages 5320–5330. 2019.

[49] T. Konstantin Rusch and Siddhartha Mishra. Coupled oscillatory recurrent neural network (co{rnn}): An accurate and (gradient) stable architecture for learning long time dependencies. In *International Conference on Learning Representations*, 2021.

[50] T Konstantin Rusch and Siddhartha Mishra. Unicornn: A recurrent model for learning very long time dependencies. *arXiv preprint arXiv:2103.05487*, 2021.

[51] Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, pages 1–13, 2019.

[52] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, undefinedukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, page 6000–6010, 2017.

[53] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[54] Winnie Xu, Ricky TQ Chen, Xuechen Li, and David Duvenaud. Infinitely deep bayesian neural networks with stochastic differential equations. *arXiv preprint arXiv:2102.06559*, 2021.

[55] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4541–4550, 2019.

[56] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[57] Amir Zeldes. The GUM corpus: Creating multilayer resources in the classroom. *Language Resources and Evaluation*, 51(3):581–612, 2017.

[58] Han Zhang, Xi Gao, Jacob Unterman, and Tom Arodz. Approximation capabilities of neural ordinary differential equations. *arXiv preprint arXiv:1907.12998*, 2019.

[59] Tianjun Zhang, Zhewei Yao, Amir Gholami, Joseph E Gonzalez, Kurt Keutzer, Michael W Mahoney, and George Biros. ANODEV2: A coupled neural ODE framework. In *Advances in Neural Information Processing Systems*, pages 5152–5162, 2019.

[60] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ODE-Net: Learning Hamiltonian dynamics with control. In *Proceedings of the International Conference on Learning Representations*, 2019.

[61] Qunxi Zhu, Yao Guo, and Wei Lin. Neural delay differential equations. In *International Conference on Learning Representations*, 2021.

[62] Juntang Zhuang, Nicha Dvornek, Xiaoxiao Li, Sekhar Tatikonda, Xenophon Papademetris, and James Duncan. Adaptive checkpoint adjoint method for gradient estimation in neural ODE. *arXiv preprint arXiv:2006.02493*, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] We provide a discussion of limitations in the conclusion.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We provide details in the SM.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]