Architecture-Aware Learning Curve Extrapolation via Graph Ordinary Differential Equation

Yanna Ding[1], Zijie Huang[2], Xiao Shou[3], Yihang Guo[2], Yizhou Sun[2], Jianxi Gao[1]
[1]Rensselaer Polytechnic Institute
[2]University of California, Los Angeles
[3]Baylor University
{dingy6, gaoj8}@rpi.edu, {zijiehuang, yihangguo, yzsun}@cs.ucla.edu, xiao_shou@baylor.edu

Abstract

Learning curve extrapolation predicts neural network performance from early training epochs and has been applied to accelerate AutoML, facilitating hyperparameter tuning and neural architecture search. However, existing methods typically model the evolution of learning curves in isolation, neglecting the impact of neural network (NN) architectures, which influence the loss landscape and learning trajectories. In this work, we explore whether incorporating neural network architecture improves learning curve modeling and how to effectively integrate this architectural information. Motivated by the dynamical system view of optimization, we propose a novel architecture-aware neural differential equation model to forecast learning curves continuously. We empirically demonstrate its ability to capture the general trend of fluctuating learning curves while quantifying uncertainty through variational parameters. Our model outperforms current state-of-the-art learning curve extrapolation methods and pure time-series modeling approaches for both MLP and CNN-based learning curves. Additionally, we explore the applicability of our method in Neural Architecture Search scenarios, such as training configuration ranking.

Introduction

Training neural architectures is a resource-intensive endeavor, often demanding considerable computational power and time. Researchers have developed various methodologies to predict the performance of neural networks early in the training process using learning curve data. Some methods (Domhan, Springenberg, and Hutter 2015; Gargiani et al. 2019; Adriaensen et al. 2023) apply Bayesian inference to project these curves forward, while others employ time-series prediction techniques, such as LSTM networks. Despite their effectiveness, these approaches (Swersky, Snoek, and Adams 2014; Baker et al. 2017) typically overlook the architectural features of networks, missing out on crucial insights that could be derived from the models' topology.

On another front, architecture-based predictive models have been developed to forecast network performance based purely on NN structures (Shi et al. 2019; Friede et al. 2019; Wen et al. 2020; Tang et al. 2020; Yan et al. 2020; Ning et al. 2020; Xu et al. 2019; Siems et al. 2020). These models facilitate a deeper understanding of the relationship between architectures and their performance. However, they are limited in their ability to predict precise learning curve values at specific epochs and struggle to capture the variability in performance that a single architecture can exhibit under diverse training conditions.

Moreover, there is a growing interest in conceptualizing the optimization process during NN training as a dynamical system. By considering the step size in gradient descent as approaching zero, it is possible to formulate an ordinary differential equation for the model parameters (Su, Boyd, and Candes 2016; Zhang, Frei, and Bartlett 2024; Maskan, Zygalakis, and Yurtsever 2023). This perspective is useful for analyzing the convergence of different optimization algorithms, especially for convex problems. Building on this foundational idea, we propose an innovative approach that models the evolution of learning curves using neural differential equations (Chen et al. 2018), tailored for a inductive setting where the trained learning curve predictor is applicable to new learning curves generated under various training configurations, such as different architectures, batch sizes, and learning rates. This approach leverages recent advancements in differential equations to provide a flexible framework capable of handling the complexities of modern neural training processes.

Our method merges the structural attributes of neural architectures with the dynamic nature of learning curves. We utilize a seq2seq variational autoencoder framework to analyze the initial stages of a learning curve and predict its future progression. This predictive capability is further enhanced by an architecture-aware component that produces a graph-level embedding from the architecture's topology, employing techniques like Graph Convolutional Networks (GCN) (Kipf and Welling 2016) and Differentiable Pooling (Ying et al. 2018). This integration not only improves the accuracy of learning curve extrapolations compared to existing methods but also significantly facilitates model ranking, potentially leading to more efficient use of computational resources, accelerated experimentation cycles, and faster progress in the field of machine learning.

Our contributions are twofold:

- We introduce an architecture-aware, dynamical system-based approach to model learning curves from different architectures for a given source task. Our model can pre-

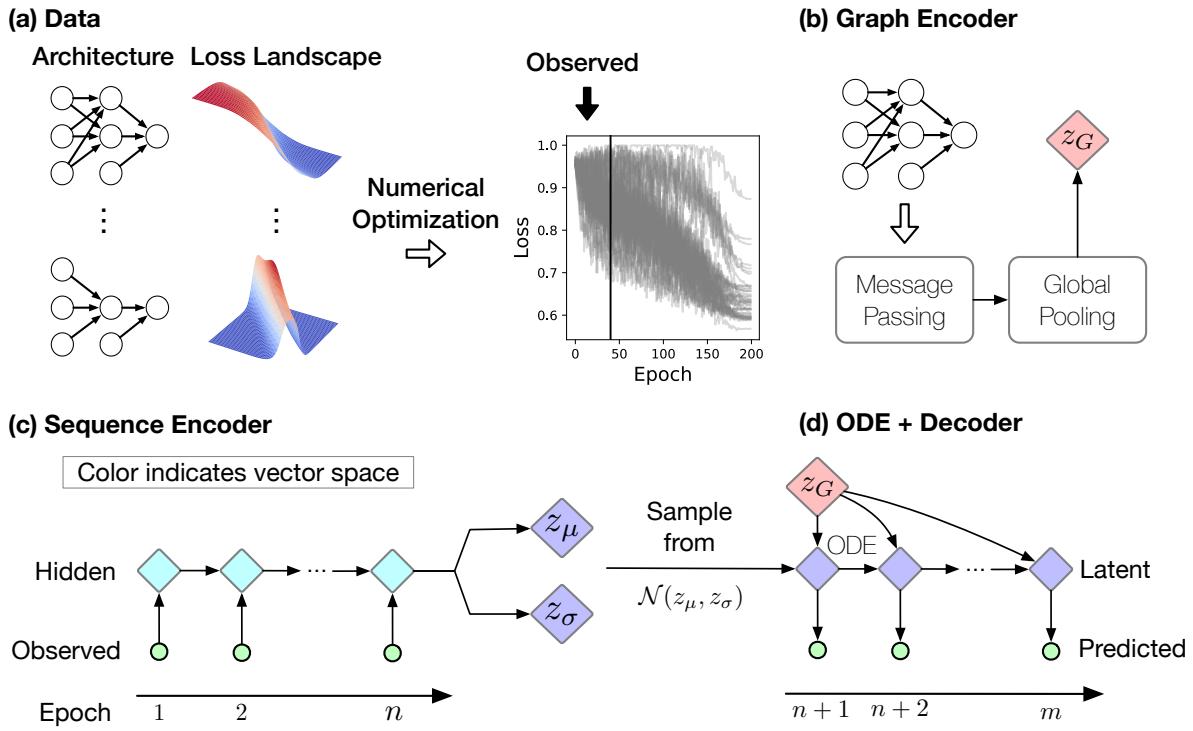


Figure 1: Overall framework. (a) Given fixed training data and a specific task, each architecture defines a unique loss landscape. We employ a numerical optimization method (e.g., gradient descent) to sample a loss trajectory across this landscape. Our dataset consists of various architectures paired with their corresponding loss curves. The model input is an observation window, which it utilizes to predict the trajectory within the subsequent prediction window. (b) Our model architecture incorporates a graph encoder that captures the architectural structure by extracting a single embedding through message passing and global pooling. (c) We initialize a latent distribution at the first epoch. A GRU unit processes this information, generating a hidden vector at each observed timestamp based on the prior hidden state and the current loss value. (d) Finally, we integrate the ODE that governs the evolution of the latent loss states, with each time step modulated by the graph embedding.

dict the learning curves of unseen architectures using only a few observed epochs.

- Our method improves model ranking by analyzing just a limited number of learning curve epochs, such as 10. This approach speeds up model selection by 20 times compared to traditional full-cycle stochastic gradient descent training.

Related Work

Learning curve extrapolation. Previous studies have explored learning curve prediction through diverse approaches (Swersky, Snoek, and Adams 2014; Domhan, Springenberg, and Hutter 2015; Baker et al. 2017; Chandrashekaran and Lane 2017; Gargiani et al. 2019; Ru et al. 2021; Klein et al. 2022; Adriaensen et al. 2023). A line of work has focused on Bayesian frameworks. Specifically, (Domhan, Springenberg, and Hutter 2015) utilized a weighted combination of functions to predict mean future validation accuracy and facilitate early termination of underperforming training runs. Building on this, (Chandrashekaran and Lane 2017) extended basis function extrapolation by incorporating historical learning curves from previous training runs, while (Klein et al. 2022) proposed a Bayesian neural network to flexibly model learning curves, removing the constraint that each epoch must outperform the previous one and thereby reducing instability in predictions. More recently, (Adriaensen et al. 2023) applied a prior-data-fitted network training paradigm to enhance sampling efficiency from the posterior distribution of learning curves. Despite these advancements, existing methods overlook the role of architectural design, whereas our work explicitly incorporates this information to better model and understand the evolution of learning curves.

Architecture-based performance prediction. Advances in Graph Neural Networks (GNNs) have led to innovative methods for predicting the performance of neural network architectures (Shi et al. 2019; Friede et al. 2019; Wen et al. 2020; Tang et al. 2020; Yan et al. 2020; Ning et al. 2020; Xu et al. 2019; Siems et al. 2020). In exploring unsupervised learning strategies, Yan et al. (Yan et al. 2020) used architecture embeddings created by a pre-trained model to feed a Gaussian Process model for performance prediction. These studies often incorporate foundational GNN models such as the GCN (Kipf and Welling 2016) and the Graph Isomor-

phism Network (GIN) (Xu et al. 2018) to effectively process input architectures. Various training objectives have been considered, including Mean Squared Error (MSE) (Shi et al. 2019; Wen et al. 2020; Ning et al. 2020; Tang et al. 2020), graph reconstruction loss (Friede et al. 2019; Tang et al. 2020; Yan et al. 2020), and pair-wise ranking loss (Ning et al. 2020; Xu et al. 2019), highlights the diverse methods aimed at improving the prediction of architecture performance. Building on these foundations, our approach goes beyond simply using the graph representation to extract a scalar performance value; instead, it integrates the graph information into the ODE of loss and can extrapolate to any time step of interest, including the value at convergence.

Dynamical system modeling. Neural ordinary differential equations (NODE) (Chen et al. 2018) introduce a general framework for parameterizing ODEs with deep neural networks, deriving backpropagation through the adjoint sensitivity method. Variational autoencoders combined with NODE, as introduced in (Rubanova, Chen, and Duvenaud 2019), are used to predict dynamics from irregularly sampled time-series data. Building on these developments, recent works (Huang, Sun, and Wang 2020, 2021; Luo et al. 2023; Huang et al. 2024b; Jiang et al. 2023; Huang et al. 2024a) further integrate graph neural networks (GNNs) with NODE to model temporal graphs, allowing for the representation of evolving nodes and edges over time. In contrast to modeling individual nodal trajectories, this work employs GNNs as a graph reduction technique, using their embeddings to drive the evolution of learning curves within the NODE framework.

Method

Problem Formulation

Learning curves, such as train or test loss, are generated by optimizing a neural network on various source tasks including adult income classification, image classification, and housing price regression (Vanschoren et al. 2014). Our goal is to train a single latent ODE model capable of extrapolating learning curves for a given source task across different architectures. The model infers full learning curves of length m using only the initial n epochs $y_1, \ldots, y_n$ and the corresponding network architecture, denoted as G.

Our approach transforms the conventional discrete optimization process into a continuous domain, operating within the continuous time interval $[0, T_{\max}]$, where 0 marks the start and $T_{\max}$ corresponds to the last epoch m. The time for each epoch, t_i, is defined as $i\Delta t$ with $\Delta t = T_{\max}/m$.

We employ a seq2seq variational autoencoder framework (Rubanova, Chen, and Duvenaud 2019; Huang, Sun, and Wang 2020, 2021), where a sequence encoder parameterized by ϕ processes the early part of the learning curve to estimate the variational parameters of the posterior distribution q_ϕ, which determines the latent state at the start of the prediction period, represented by $\boldsymbol{z}_{n+1} \in \mathbb{R}^D$. A numerical solver then integrates an ODE function, denoted as $f : \mathbb{R}^D \to \mathbb{R}^D$, governing this latent state, starting from the initial condition $\boldsymbol{z}_{n+1}$. Each subsequent latent state $\boldsymbol{z}_i$ is decoded independently to predict the output $\hat{y}_i$ for $i > n$. This process is mathematically represented as follows:

$$\boldsymbol{z}_{n+1} \sim q_\phi(\boldsymbol{z}_{n+1} | \{y_i, t_i\}_{i=1}^n) \tag{1}$$

$$\boldsymbol{z}_{n+1}, \ldots, \boldsymbol{z}_m = \mathrm{ODESolve}(f, \boldsymbol{z}_{n+1}, (t_{n+1}, \ldots, t_m), G) \tag{2}$$

$$\hat{y}_i = \mathrm{Decoder}(\boldsymbol{z}_i) \quad \text{for } i > n \tag{3}$$

Here $\mathrm{ODESolve}(\cdot)$ simulates the ODE function $f(\cdot)$ to compute the latent states at time steps t_i ($i \geq n+1$), given initial condition $\boldsymbol{z}_{n+1}$ and the neural architecture G. $\mathrm{Decoder}(\cdot)$ is a neural network mapping $\boldsymbol{z}_i$ to the corresponding output $\hat{y}_i$. Since the model has access to only partial information about the optimization process, it cannot fully determine the loss landscape or accurately trace the trajectory within it. Therefore, we employ a variational framework to quantify uncertainty by estimating the most probable values and their variability. We refer to our method as Learning Curve GraphODE (LC-GODE), highlighting the integration of architectures within the ODE framework to model learning curves. The illustration of our approach is shown in Figure 1.

Architecture-aware Differential Equation

Observed time series encoder. We use a sequence encoder to compute the mean and standard deviation of the posterior distribution $q_\phi(\boldsymbol{z}_{n+1} | \{y_i, t_i\}_{i=1}^n)$, which is assumed to be Gaussian:

$$\boldsymbol{z}_{n+1} \sim q_\phi(\boldsymbol{z}_{n+1} | \{y_i, t_i\}_{i=1}^n) = \mathcal{N}(\mu_{\boldsymbol{z}_{n+1}}, \sigma_{\boldsymbol{z}_{n+1}}) \tag{4}$$

$$\mu_{\boldsymbol{z}_{n+1}}, \sigma_{\boldsymbol{z}_{n+1}} = \mathrm{SeqEncoder}_\phi(\{y_i, t_i\}_{i=1}^n) \tag{5}$$

We implement this using an RNN with GRU units for the sequence encoder. Other encoder options include Self-Attention (Vaswani et al. 2017) and Temporal Convolutional Networks (TCN) (Pandey and Wang 2019), which adapt well to varying observation lengths. We conduct an ablation study in the experimental section to evaluate the performance of these alternative sequence encoder implementations.

Architecture encoder. Assuming the graph representation G contains N nodes, the adjacency matrix $A \in \mathbb{R}_{\geq 0}^{N \times N}$ details node connections, where $A_{ij} > 0$ represents the edge from node i to node j. We analyze two foundational neural network types: MLPs and CNNs. In MLPs, nodes correspond to neurons and edges correspond to the presence of the connections between neurons, whereas in CNNs, nodes represent feature maps and edges depict operations like 1×1 and 3×3 convolutions, or 3×3 average pooling. For CNNs, we adopt the cell-based representation (Dong and Yang 2020; Liu, Simonyan, and Yang 2018), which consists of four principal building blocks: stems, normal cells, reduction cells, and classification heads. The stem block is a fixed sequence of convolutional layers to process the input images. This is followed typically by 14-20 cells with reduction cells placed at 1/3 and 2/3 of the total depth. A normal cell contains 4 nodes, each of which belongs to the set of operations: skip connections, identity or zero (indicating the presence or absence of connections between certain layers), 1×1 and 3×3 convolutions, and 3×3 average pooling. Finally, the classification head employs a global pooling layer followed by a single fully connected layer and returns the network's output.

Since the cell is repeated throughout this macro-skeleton, a CNN can be uniquely represented by its cell.

To derive a graph-level embedding, we first implement node-level message passing, followed by global pooling to extract a global embedding. Our method differs from existing architecture-based performance prediction approaches (Liu, Simonyan, and Yang 2018; Wen et al. 2020; Knyazev et al. 2021) by treating the node as a feature map rather than focusing on operations. Nodal features are calculated from the in-degree and out-degree of each node, normalized by the total number of edges. For MLPs, edge features are binary, while for CNN cells, they are integers representing operation types: $\{\text{zeroize} : 0, 1 \times 1 \text{ conv} : 1, 3 \times 3 \text{ conv} : 2, 3 \times 3 \text{ avg pooling} : 3\}$. The nodal feature matrix is denoted as $X \in \mathbb{R}^{N \times d}$. The graph encoding process returns a vector representation of the architecture.

$$\boldsymbol{z}_G = \text{ArchEncoder}_{\theta_1}(X). \tag{6}$$

For node-level message passing, we employ GCN layers and normalize the adjacency matrix to stabilize training, following (Kipf and Welling 2016).

$$\tilde{A} = A + I, \quad \tilde{D} = \sum_j \tilde{A}_{ij}.$$

A GCN layer then transforms the nodal features into a hidden representation:

$$Z = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}XW \tag{7}$$

where $W \in \mathbb{R}^{D \times d}$ is the feature transformation matrix. Applying l GCN layers aggregates information from l-hop neighbors. We employ learnable pooling, among other methods such as average- and max-pooling, and investigate each in our ablation study. The graph embedding $\boldsymbol{z}_G$ contributes to the evolution of the latent state, as detailed in the next section, which introduces the latent ODE.

Latent ordinary differential equation. The transformation from discrete to continuous domain is predicated on the assumption that the step size, or learning rate, approaches zero, thereby approximating the time derivative of NN parameters Θ using $\frac{d\Theta}{dt} = -\frac{\partial \mathcal{L}}{\partial \Theta}$ (Su, Boyd, and Candes 2016), where $\mathcal{L}$, Θ denotes the objective function and parameters from the source task. Assuming the parameters implicitly depends on time t, the time derivative of the training loss can be written as:

$$\frac{d\mathcal{L}(\Theta(t))}{dt} = \frac{\partial \mathcal{L}}{\partial \Theta}^\top \frac{d\Theta}{dt} = -\left(\frac{\partial \mathcal{L}}{\partial \Theta}\right)^\top \left(\frac{\partial \mathcal{L}}{\partial \Theta}\right) \tag{8}$$

The ground truth ODE for loss is independent of time. Therefore we adopt autonomous differential equations to describe the continuous evolution of the learning curves. On the other hand, we do not directly use the exact formula, as this involves the computation of backpropagation of the source task, which is potentially computation intensive. Moreover, our primary goal is not to derive exact ODEs for each training configuration. Instead, we focus on efficiently inferring learning curves for new training configurations. To achieve this, we leverage the latent space, using the expressivity of hidden neurons to capture common patterns across learning curves from the same source task, despite variations in underlying architectures and hyperparameters.

Given that the training data for the source task remains fixed, the loss landscape varies depending on the architecture. Therefore, it's sufficient to model a universal latent ODE as a function of the architecture, enabling it to describe the evolution of various learning curves, each corresponding to a different architecture. Our latent ODE is formalized by the equation:

$$\dot{\boldsymbol{z}} = f_{\theta_2}([\boldsymbol{z}||\boldsymbol{z}_G]) \tag{9}$$

Here, $\dot{\boldsymbol{z}}$ denotes the derivative of the latent state vector $\boldsymbol{z}$ with respect to time, driven by the function f_{θ_2}, which takes as input both the latent state $\boldsymbol{z}$ and a graph-level embedding $\boldsymbol{z}_G$. This combination allows the model to simultaneously consider the dynamic properties of the learning curve and the static characteristics of the architecture, enhancing the predictive capability of the system. Eq (9) can be regarded as a single-agent representation of the dynamics induced by NN training, which involves multiple trajectories of neurons, edge weights, and loss. This reduced representation of coupled dynamical system has been explored in (Gao, Barzel, and Barabási 2016; Laurence et al. 2019) to study the tipping point of the original network dynamics. The difference from the prior dynamical system reduction approach is that the graph reduction mechanism is learnable so that the model can be adapted to unseen trajectories derived from different optimization trials.

Finally, numerical integration of Eq (9) yields a time series of latent states $\boldsymbol{z}_i$ $(n < i \leq m)$. Each latent state is independently decoded by a function $\hat{y}_i = \text{Decoder}_{\theta_3}(\boldsymbol{z}_i)$.

Training objective. To optimize our model, we maximize the evidence lower bound (ELBO), fomulated as follows

$$\begin{aligned}\text{ELBO}(\phi, \theta) = \mathbb{E}_{\boldsymbol{z}_{n+1}}[\log p_\theta(y_{n+1}, \ldots, y_m)] \\ - \text{KL}[q_\phi(\boldsymbol{z}_{n+1}|\{y_i, t_i\})_{i=1}^n || p(\boldsymbol{z}_{n+1})]\end{aligned} \tag{10}$$

The first term in the ELBO equation represents the expected log-likelihood of observing future outputs given the latent states, as parameterized by $\theta = (\theta_1, \theta_2, \theta_3)$, where θ_1, θ_2, and θ_3 correspond to the architecture encoder, the ODE function, and the decoder, respectively. The second term penalizes the divergence (measured by the KL-divergence) between the posterior distribution of the latent states and their prior distribution, enforcing a regularization that anchors the posterior closer to the prior. This balance ensures that while the model remains flexible enough to capture complex patterns in data, it also maintains a level of generalization that prevents overfitting.

Scalability and computational cost. The computational cost of numerical integration is minimal because the ODE models only the evolution of the loss embedding with dimension D, rather than modeling each node in the architecture individually. The runtime of the forward pass is bounded by $O(D^2T)$, where $T = m - n$ is the number of integration time steps, assuming the ODE function is implemented as

an MLP with a fixed number of layers. Consequently, the runtime of the ODE component remains independent of the overall size of the neural network.

Experiments

We showcase LC-GODE's ability to forecast model performance across diverse AutoML benchmarks. First, we compare it to six learning curve extrapolation methods on real-world datasets using stochastic gradient descent for tabular and image tasks, training each source task separately. Next, we evaluate its effectiveness in ranking training configurations by predicted optimal performance. Finally, we analyze model sensitivity to architecture variants, time-series encoders, and hyperparameters. Our code and supplementary materials are publicly available[1].

Datasets. We consider test loss and test accuracy curves for both MLP-based and CNN-based architectures. Specifically for MLPs, we use `car` and `segment` tabular data binary classification tasks from OpenML (Vanschoren et al. 2014) as source tasks. Following LCBench (Zimmer, Lindauer, and Hutter 2021), we randomly generate training configurations that include variables such as the number of layers, number of hidden units, and learning rates. For each dataset, we conduct 550 optimization trials across 200 epochs. For CNN-based models, we employ the NAS-Bench-201 dataset (Dong and Yang 2020), which provides comprehensive learning curves for each architecture over a span of 200 epochs across two image datasets: CIFAR-10, CIFAR-100 (Krizhevsky, Hinton et al. 2009). We randomly select 5,000 architectures from CIFAR-10 and CIFAR-100 to form our dataset. Furthermore, we reserve 20% of all trials as the test set for each MLP source task and 25% for each CNN source task.

Extrapolating Real-world Learning Curves

Experimental setup. The goal of this experiment is to evaluate the LC-GODE model against established learning curve prediction methods using real-world benchmarks. We train our model separately on the test loss curves of each source task. The condition length is set to 10 epochs for all methods in this experiment. The instantiation of LC-GODE that we report features: (i) an architecture encoder that utilizes 2 layers and employs a learnable pooling technique, (ii) an observed time-series encoder implemented using GRU, (iii) an ODE function with a 2-layer MLP and integrated using the Runge-Kutta 4 method (Butcher 1996). Further details on the evaluation metrics and the training settings can be found in the supplemental materials.

Baselines. We evaluate our model against six methods, including three Bayesian approaches and three general time-series prediction approaches. (i) LC-BNN: This method utilizes a Bayesian neural network to model the posterior distribution of future learning curves. The probability function is constructed from a combination of basis functions, following the approach described by Domhan et al. (Domhan, Springenberg, and Hutter 2015). (ii) LC-PFN: A transformer-based model is trained on synthetic curves that are generated from a pre-defined prior. This method serves as an efficient alternative to traditional Markov Chain Monte-Carlo (MCMC) techniques for sampling from posterior distributions. (iii) VRNN: A probabilistic model utilizing random forests and Bayesian recurrent neural networks. (iv) LSTM: This Recurrent Neural Network captures sequential dependencies. Before the observation cutoff at n, input states are derived from actual observations. Post n, the model uses its own predictions from previous timestamps as inputs. (v) NODE: Latent Ordinary Differential Equations, focusing primarily on modeling the latent loss representation without incorporating architectural information. (vi) NSDE: Latent Stochastic Differential Equations, consisting of a drift term and a diffusion term. The drift term is the same as NODE, and the sequence encoder and decoder is the same as in our model. Both NODE and NSDE are trained using the variational framework.

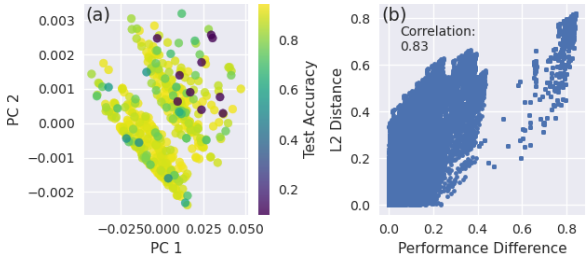


Figure 2: (a) Graph embedding projections. (b) Pairwise initial latent embedding distance vs performance difference.

Results. We evaluate performance using Mean Absolute Percentage Error (MAPE) and Root Mean Squared Error (RMSE) with different prediction lengths. Table 1 shows the extrapolation error for test accuracy curves from four datasets. Due to space constraints, the comparison results for loss curve extrapolation are provided in the supplement. Our proposed LC-GODE outperforms all baselines on these datasets. Specifically, LC-GODE reduces the error on test accuracy curves by 36.13%, 30.72%, 34.97%, and 59.63%, and on test loss curves by 65.5%, 44.61%, 20.1%, and 23.45% compared to the NODE model without architecture information. This improvement is due to the incorporation of architecture information with graph embedding. The architectures with similar performance are mapped to nearby locations in the hidden space, as shown in Figure 2(a). To demonstrate the advantage of jointly training the time-series encoder with the graph encoder, we compare the optimal performance difference of two configurations with the distance between their initial latent states in Figure 2(b). When architecture information is included, the correlation between these distances increases from 0.77 to 0.83 for the CIFAR-10 test accuracy curves. For a detailed comparison over epochs, we plot MAPE at 10 prediction lengths for NODE, NSDE, and LC-GODE on both test accuracy and loss curves (Figure 3). Overall, models perform better and have larger improvement on test accuracy curves compared to loss curves. This could be attributed to the fact that classification accuracy is less sensitive to decision boundary changes than cross-entropy loss, making it

[1]https://github.com/dingyanna/LC-GODE.git

	car						segment					
	MAPE			RMSE			MAPE			RMSE		
Epochs	80	140	200	80	140	200	80	140	200	80	140	200
LC-BNN	0.5487	0.4887	0.4534	0.4232	0.3838	0.3592	0.6211	0.5688	0.5370	0.5384	0.4960	0.4694
LC-PFN	0.0598	0.0681	0.0723	0.0443	0.0502	0.0528	0.0654	0.0708	0.0729	0.0497	0.0543	0.0555
VRNN	0.1857	0.1923	0.1923	0.1514	0.1511	0.1511	0.1840	0.1742	0.1742	0.1613	0.1557	0.1557
LSTM	0.0853	0.1104	0.1251	0.0561	0.0709	0.0790	0.0825	0.1126	0.1346	0.0478	0.0640	0.0758
NODE	0.0683	0.0730	0.0764	0.0459	0.0499	0.0531	0.0794	0.0837	0.0853	0.0574	0.0597	0.0609
NSDE	0.0751	0.0768	0.0779	0.0503	0.0515	0.0522	0.0817	0.0854	0.0864	0.0595	0.0610	0.0614
LC-GODE	**0.0431**	**0.0463**	**0.0488**	**0.0328**	**0.0349**	**0.0365**	**0.0566**	**0.0591**	**0.0608**	**0.0462**	**0.0477**	**0.0487**
	cifar10						cifar100					
	MAPE			RMSE			MAPE			RMSE		
Epochs	80	140	200	80	140	200	80	140	200	80	140	200
LC-BNN	0.4235	0.4204	0.4036	0.2101	0.2116	0.2260	1.2441	1.4087	1.2946	0.1161	0.1219	0.1373
LC-PFN	0.1514	0.1618	0.1782	0.0835	0.1023	0.1325	0.3115	0.3536	0.3749	0.0738	0.1062	0.1549
VRNN	0.1231	0.1316	0.1419	0.0905	0.0917	0.0987	0.3621	0.5105	0.5801	0.1153	0.1161	0.1224
LSTM	0.1391	0.1467	0.1309	0.0752	0.0736	0.0682	0.3853	0.5599	0.5761	0.0659	0.0715	0.0720
NODE	0.1457	0.1525	0.1404	0.0711	0.0717	0.0727	0.4895	0.6593	0.6592	0.0681	0.0760	0.0798
NSDE	0.1491	0.1454	0.1245	0.0686	0.0670	0.0645	0.4547	0.5076	0.4201	0.0629	0.0669	0.0665
LC-GODE	**0.1107**	**0.1093**	**0.0913**	**0.0645**	**0.0603**	**0.0557**	**0.2708**	**0.3205**	**0.2661**	**0.0621**	**0.0636**	**0.0625**

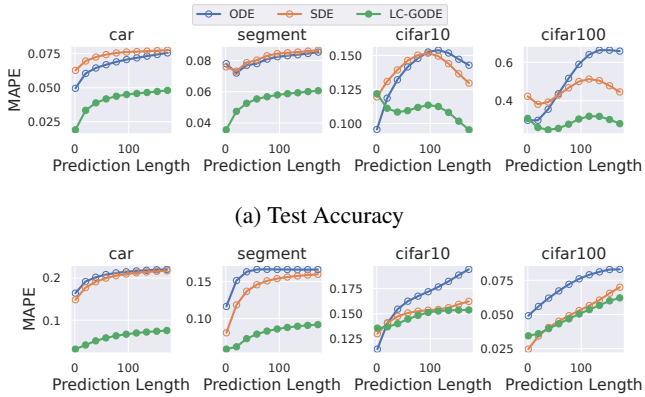
Table 1: Extrapolation error for test accuracy curves derived from 2 tabular tasks and 2 image classification tasks. The percentage represents the fraction of the entire prediction window over which the error is computed and averaged. Green color highlights our approach. Bold denotes the best model and underline denotes the second best model.

less variable and easier to predict.

Regarding the baseline comparisons, LC-PFN emerges as the second most effective approach for extrapolating test accuracy curves of the car and segment source tasks, while the NSDE model ranks as the second in predicting test loss curves for CIFAR-10 and CIFAR-100. Both NSDE and NODE models demonstrate comparable performance, with NSDE marginally outperforming NODE. These results underscore the viability of employing time-series approaches for addressing learning curve extrapolation challenges. Notably, the slight advantage of NSDE over NODE suggests subtle benefits in capturing stochastic dynamics that may be present in complex learning scenarios. This comparison highlights the potential for refined time-series models to enhance predictive accuracy and adaptability in diverse training environments.

Model ranking We evaluate our method's efficacy in ranking training configurations by comparing the predicted and true optimal performances at a fixed snapshot, employing two metrics: *regret* (the performance discrepancy between the actual and predicted best configurations) and *ranking* (the position of the predicted best configuration according to the true learning curves). These metrics demonstrate whether the predicted performance can effectively guide the selection of a performant configuration.

As shown in Table 2, our proposed approach, LC-GODE, enhances the *ranking* of the predicted best model by 3 positions on the segment dataset and by 8 positions on the CIFAR-10 dataset, and reduces *regret* by 96% on CIFAR-10 compared to the superior baseline among NODE and NSDE for test accuracy curves. Nevertheless, when employing test



(a) Test Accuracy

(b) Test Loss

Figure 3: Extrapolation error (MAPE) for learning curves w.r.t. prediction length.

loss to identify the best model, the overall *ranking* for all methods declines, indicating that predicted test loss is less effective for model selection.

Our approach identified a performant model with a 20x speedup compared to exhaustive brute-force training using stochastic gradient descent (SGD) on the MLP datasets. The speedup is computed as the total runtime needed to fully train all configurations via SGD, divided by the sum of the actual training time for n epochs and the inference time required by LC-GODE. The inference time for LC-GODE is approximately 0.8 seconds, and the majority of the model selection time is spent on the initial n epochs of SGD training.

Metric	Dataset	Accuracy			Loss		
		NODE	NSDE	LC-GODE	NODE	NSDE	LC-GODE
regret	car	0.0023	0.0023	0.0023	0.4570	0.0950	0.0000
	segment	0.0052	0.0017	0.0017	0.0100	0.0100	0.0100
	cifar10	0.0025	0.0101	0.0004	0.0164	0.0243	0.0048
	cifar100	0.0000	0.0374	0.0000	0.0262	0.0606	0.0303
ranking	car (110)	2	2	2	43	4	1
	segment (110)	18	7	4	2	2	2
	cifar10 (1250)	10	86	2	21	73	3
	cifar100 (1250)	1	142	1	87	607	137

Table 2: Model selection evaluation based on *regret* and the *ranking* of the predicted best model. The number in the bracket after the dataset denotes the total number of configurations.

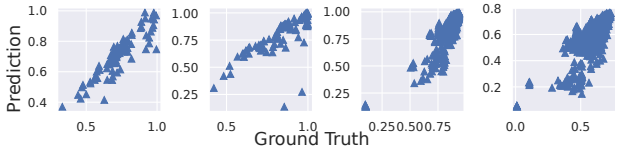
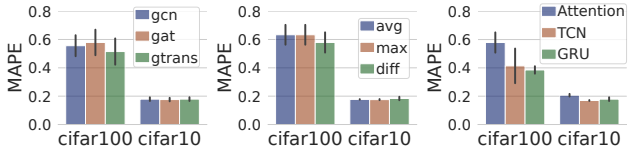


Figure 4: True vs predicted best test accuracy for car, segment, CIFAR-10, CIFAR-100, respectively.



(a) Message passing (b) Graph pooling (c) Sequence encoder

Figure 5: Ablation study for encoders.

Additional results on the training and inference runtime of all methods are provided in the supplement.

Figure 4 further shows the true vs predicted best test accuracy for the 2 tabular data and 2 image classification source tasks, further indicating a high correlation between predicted and true metrics.

Ablation Study

We explore three variations for each of the following components: the message passing mechanism, graph pooling, and time series encoder. The message passing mechanisms include GCN (Kipf and Welling 2016), Graph Attention Networks (GAT) (Veličković et al. 2017), and Graph Transformers (Hu et al. 2020). For pooling methods, we use average pooling, max-pooling, and a learnable pooling method based on DiffPool (Ying et al. 2018), where a separate GCN module determines each node's contribution to the global embedding. The time-series encoder variations include self-attention (Vaswani et al. 2017), TCN (Pandey and Wang 2019), and an autoregressive version of GRU (Dey and Salem 2017). The model is trained using early stopping if no improvement is observed after 50 epochs.

As shown in Figure 5, for CIFAR-100, the combination of graph transformers, DiffPool, and GRU achieved the best results, while for CIFAR-10, GAT, max-pooling, and TCN performed better. This suggests that different combinations of encoders can affect extrapolation accuracy, with a more significant impact observed on CIFAR-100.

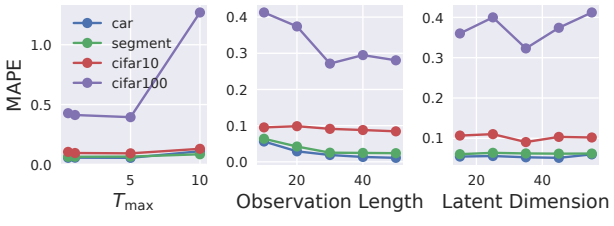


Figure 6: Hyperparameter sensitivity study.

Hyperparameter sensitivity. We analyze the impact of three hyperparameters: maximal time $T_{\max}$, observation length, and hidden dimension (Figure 6). A larger $T_{\max}$ negatively impacts performance. As observation length increases, the model receives more information about the curve, requiring less extrapolation and thus reducing error. The performance remains relatively constant when the latent dimension is within the range of 20 to 50.

Conclusion

In this work, we introduced LC-GODE, a novel approach that integrates architectural insights with learning curve extrapolation from a dynamical systems perspective. Using early performance data, it predicts future learning trajectories, improving test loss and accuracy forecasts. This architecture-aware method outperforms existing extrapolation techniques, enabling better model ranking and selection with minimal epochs. It achieves a 20× speedup in model selection over full training cycles with stochastic gradient descent. Future work may enhance generalization by incorporating source data impact across tasks.

Acknowledgements

Y.N.D. and J.X.G. are supported by the National Science Foundation (No. 2047488), and by the Rensselaer-IBM AI Research Collaboration. This work was partially supported by NSF 2211557, NSF 2119643, NSF 2303037, NSF 2312501, NASA, SRC JUMP 2.0 Center, Amazon Research Awards, and Snapchat Gifts.

References

Adriaensen, S.; Rakotoarison, H.; Müller, S.; and Hutter, F. 2023. Efficient Bayesian Learning Curve Extrapolation using Prior-Data Fitted Networks. In Oh, A.; Naumann, T.; Globerson, A.; Saenko, K.; Hardt, M.; and Levine, S., eds., *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Baker, B.; Gupta, O.; Raskar, R.; and Naik, N. 2017. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823*.

Butcher, J. C. 1996. A history of Runge-Kutta methods. *Applied numerical mathematics*, 20(3): 247–260.

Chandrashekaran, A.; and Lane, I. R. 2017. Speeding up hyper-parameter optimization by extrapolation of learning curves using previous builds. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2017, Skopje, Macedonia, September 18–22, 2017, Proceedings, Part I 10*, 477–492. Springer.

Chen, R. T.; Rubanova, Y.; Bettencourt, J.; and Duvenaud, D. K. 2018. Neural ordinary differential equations. *Advances in neural information processing systems*, 31.

Dey, R.; and Salem, F. M. 2017. Gate-variants of gated recurrent unit (GRU) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, 1597–1600. IEEE.

Domhan, T.; Springenberg, J. T.; and Hutter, F. 2015. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-fourth international joint conference on artificial intelligence*.

Dong, X.; and Yang, Y. 2020. Nas-bench-201: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326*.

Friede, D.; Lukasik, J.; Stuckenschmidt, H.; and Keuper, M. 2019. A variational-sequential graph autoencoder for neural architecture performance prediction. *arXiv preprint arXiv:1912.05317*.

Gao, J.; Barzel, B.; and Barabási, A.-L. 2016. Universal resilience patterns in complex networks. *Nature*, 530(7590): 307–312.

Gargiani, M.; Klein, A.; Falkner, S.; and Hutter, F. 2019. Probabilistic rollouts for learning curve extrapolation across hyperparameter settings. *arXiv preprint arXiv:1910.04522*.

Hu, Z.; Dong, Y.; Wang, K.; and Sun, Y. 2020. Heterogeneous graph transformer. In *Proceedings of the web conference 2020*, 2704–2710.

Huang, Z.; Hwang, J.; Zhang, J.; Baik, J.; Zhang, W.; Wodarz, D.; Sun, Y.; Gu, Q.; and Wang, W. 2024a. Causal Graph ODE: Continuous Treatment Effect Modeling in Multi-agent Dynamical Systems. In *Proceedings of the ACM Web Conference 2024*, WWW '24, 4607–4617.

Huang, Z.; Sun, Y.; and Wang, W. 2020. Learning continuous system dynamics from irregularly-sampled partial observations. *Advances in Neural Information Processing Systems*, 33: 16177–16187.

Huang, Z.; Sun, Y.; and Wang, W. 2021. Coupled graph ode for learning interacting system dynamics. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 705–715.

Huang, Z.; Zhao, W.; Gao, J.; Hu, Z.; Luo, X.; Cao, Y.; Chen, Y.; Sun, Y.; and Wang, W. 2024b. Physics-Informed Regularization for Domain-Agnostic Dynamical System Modeling. *Advances in Neural Information Processing Systems*.

Jiang, S.; Huang, Z.; Luo, X.; and Sun, Y. 2023. CF-GODE: Continuous-Time Causal Inference for Multi-Agent Dynamical Systems. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

Kipf, T. N.; and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Klein, A.; Falkner, S.; Springenberg, J. T.; and Hutter, F. 2022. Learning curve prediction with Bayesian neural networks. In *International conference on learning representations*.

Knyazev, B.; Drozdzal, M.; Taylor, G. W.; and Romero Soriano, A. 2021. Parameter prediction for unseen deep architectures. *Advances in Neural Information Processing Systems*, 34: 29433–29448.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning multiple layers of features from tiny images.

Laurence, E.; Doyon, N.; Dubé, L. J.; and Desrosiers, P. 2019. Spectral dimension reduction of complex dynamical networks. *Physical Review X*, 9(1): 011042.

Liu, H.; Simonyan, K.; and Yang, Y. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

Luo, X.; Yuan, J.; Huang, Z.; Jiang, H.; Qin, Y.; Ju, W.; Zhang, M.; and Sun, Y. 2023. HOPE: High-order Graph ODE For Modeling Interacting Dynamics.

Maskan, H.; Zygalakis, K.; and Yurtsever, A. 2023. A Variational Perspective on High-Resolution ODEs. In Oh, A.; Naumann, T.; Globerson, A.; Saenko, K.; Hardt, M.; and Levine, S., eds., *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Ning, X.; Zheng, Y.; Zhao, T.; Wang, Y.; and Yang, H. 2020. A generic graph-based neural architecture encoding scheme for predictor-based nas. In *European Conference on Computer Vision*, 189–204. Springer.

Pandey, A.; and Wang, D. 2019. TCNN: Temporal convolutional neural network for real-time speech enhancement in the time domain. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 6875–6879. IEEE.

Ru, R.; Lyle, C.; Schut, L.; Fil, M.; van der Wilk, M.; and Gal, Y. 2021. Speedy performance estimation for neural architecture search. *Advances in Neural Information Processing Systems*, 34: 4079–4092.

Rubanova, Y.; Chen, R. T.; and Duvenaud, D. K. 2019. Latent ordinary differential equations for irregularly-sampled time series. *Advances in neural information processing systems*, 32.

Shi, H.; Pi, R.; Xu, H.; Li, Z.; Kwok, J. T.; and Zhang, T. 2019. Multi-objective neural architecture search via predictive network performance optimization.

Siems, J.; Zimmer, L.; Zela, A.; Lukasik, J.; Keuper, M.; and Hutter, F. 2020. Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. *arXiv preprint arXiv:2008.09777*, 4: 14.

Su, W.; Boyd, S.; and Candes, E. J. 2016. A differential equation for modeling Nesterov's accelerated gradient method: Theory and insights. *Journal of Machine Learning Research*, 17(153): 1–43.

Swersky, K.; Snoek, J.; and Adams, R. P. 2014. Freeze-thaw Bayesian optimization. *arXiv preprint arXiv:1406.3896*.

Tang, Y.; Wang, Y.; Xu, Y.; Chen, H.; Shi, B.; Xu, C.; Xu, C.; Tian, Q.; and Xu, C. 2020. A semi-supervised assessor of neural architectures. In *proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 1810–1819.

Vanschoren, J.; Van Rijn, J. N.; Bischl, B.; and Torgo, L. 2014. OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2): 49–60.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Wen, W.; Liu, H.; Chen, Y.; Li, H.; Bender, G.; and Kindermans, P.-J. 2020. Neural predictor for neural architecture search. In *European conference on computer vision*, 660–676. Springer.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.

Xu, Y.; Wang, Y.; Han, K.; Chen, H.; Tang, Y.; Jui, S.; Xu, C.; Tian, Q.; and Xu, C. 2019. Rnas: Architecture ranking for powerful networks. *arXiv preprint arXiv:1910.01523*.

Yan, S.; Zheng, Y.; Ao, W.; Zeng, X.; and Zhang, M. 2020. Does unsupervised architecture representation learning help neural architecture search? *Advances in neural information processing systems*, 33: 12486–12498.

Ying, Z.; You, J.; Morris, C.; Ren, X.; Hamilton, W.; and Leskovec, J. 2018. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31.

Zhang, R.; Frei, S.; and Bartlett, P. L. 2024. Trained Transformers Learn Linear Models In-Context. *J. Mach. Learn. Res.*, 25: 49:1–49:55.

Zimmer, L.; Lindauer, M.; and Hutter, F. 2021. Auto-pytorch: Multi-fidelity metalearning for efficient and robust autodl. *IEEE transactions on pattern analysis and machine intelligence*, 43(9): 3079–3090.