Supplemental Materials for Architecture-Aware Learning Curve Extrapolation via Graph Ordinary Differential Equation

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

Experimental Details

Datasets. To generate MLP learning curves, we selected 2 tabular data classification tasks from OpenML (Vanschoren et al. 2014) as detailed in Table 2. We adhere to the standard procedure outlined in LCBench (Zimmer, Lindauer, and Hutter 2021), with the exception that we introduce variability by randomly sampling the number of hidden units for each layer, rather than sampling only the maximal number of hidden units. All optimization runs are performed using SGD with decaying learning rate by 0.5 every 80 epochs. The total number of epochs is 200. We discard runs The parameters randomly sampled include four floating-point values and $2 + l$ integers, where l denotes the number of layers.

Table 1: Hyperparameter setting used to generate MLP data.

Hyperparameter	**Value**	**Log-scale**
Batch size	$[16, 512]$	Y
Learning rate	$[0.0001, 0.1]$	Y
Batch size	$[16, 512]$	Y
Weight decay	$[0.00001, 0.1]$	N
Number of layers	$[1, 5]$	N
Number of units per layer	$[16, 1024]$	Y
Dropout	$[0.0, 1.0]$	N

Table 2: Tabular datasets used to generate MLP learning curves.

Tabular dataset	**# train samples**	**# test samples**	**# features**	**# labels**
car	1296	432	6	2
segment	1732	578	19	2

Metrics. We adopt six metrics to evaluate our approach from three dimensions.

- Trajectory reconstruction. To evaluate curve reconstruction error, we utilize MAPE. Let N_{pred} denote the prediction length. Let $\hat{x}^{(i)} \in \mathbb{R}^{N_{pred}}$ denote the ith trajectory in the test set. Let $y^{(i)} \in \mathbb{R}^{N_{pred}}$ denote the ground truth trajectory. The reconstruction error for one trajectory is comuted as

$$\text{MAPE}_i = \frac{1}{N_{pred}} \sum_{t=1}^{N_{pred}} \left| \frac{y_t^{(i)} - \hat{x}_t^{(i)}}{y_t^{(i)}} \right| \tag{1}$$

The error metric for the entire test dataset is the corresponding average over all trajectories.

$$\text{MAPE} = \frac{1}{N_{traj}} \text{MAPE}_i \tag{2}$$

- Model Selection. We use Pearson correlation as adopted in [?] and Kendall τ [NasWOT].
- Efficiency. We report the training runtime per epoch and the the wall-clock time to perform inference for one architecture using our model. The speedup is computed as

$$\text{Speedup} = \frac{\text{Runtime for SGD over 52 Epochs}}{\text{Runtime for SGD over } T_{cond} \text{ epochs} + \text{LC-ODE Forward Runtime}} \tag{3}$$

Hyperparameter Setting. Without further specification, the hyperparameters to train both our model and the baselines are set according to Table 3

Table 3: Hyperparameter setting used throughout the paper.

Hyperparameter	**Value**
Latent Dimension	16
Learning Rate	0.001
Batch Size	128 (CNN) 40 (MLP)
Optimizer	AdamW (Loshchilov and Hutter 2017)
Number of Epochs	400
Condition Length	20% T_{max}
Prediction Length	80% T_{max}

Baseline Configurations.

LC-BNN. LC-BNN is a function that maps a tuple containing a configuration and an epoch $(\text{configuration}, \text{epoch})$ to the loss value associated with that configuration at the specified epoch. We represent the trajectory data as pairs of input and target values: $((\text{config}_i, t_j), y_i(t_j))$ where config_i denotes the ith configuration, t_j represents an epoch, and $y_i(t_j)$ is the corresponding metric value. For MLP configurations, we use the hyperparameters specified in LCBench, detailed in Table 1. In the case of NAS-Bench-201, we utilize the hyperparameters outlined in their respective publication. Note that all architectures within NAS-Bench-201 utilize a uniform set of hyperparameters. Nevertheless, we include these hyperparameters as input to LC-BNN, as it requires at least one additional input beyond the epoch number.

Our dataset is structured within an inductive learning framework, comprising multiple training trajectories, with the test set including unseen trajectories. Since LC-BNN does not have an associated conditional length, to ensure a fair comparison, we incorporate the conditional window from our test set into the training data for LC-BNN.

LC-PFN. To apply LC-PFN, we preprocess our data using the normalization procedure outlined in Appendix A by (Adriaensen et al. 2023). This approach ensures that our data is consistently formatted and scaled according to the specified guidelines. The normalization process transforms observed curves into a constrained range $[0, u]$ where $u \approx 1$. LC-PFN takes as input the normalized observations and outputs inferred loss values. These values are then mapped back to their original space using the inverse of the normalization function. The parameters of the normalization function are defined as $\boldsymbol{\lambda} = (\text{minimize}, l_{\text{hard}}, u_{\text{hard}}, l_{\text{soft}}, u_{\text{soft}})$:

- **min?**: A Boolean indicating whether the curve is to be minimized or maximized.
- $l_{\text{hard}}, u_{\text{hard}}$: Hard bounds defining the absolute limits of the learning curve.
- $l_{\text{soft}}, u_{\text{soft}}$: Soft bounds that guide the behavior of the learning curve.

The normalization function is formulated as:

$$g_{\boldsymbol{\lambda}}(x) = \text{cr}_{0.5}\left(\frac{c}{1 + e^{-(a(x-b))}} + d\right)$$

where:

$$\begin{aligned}
cr_{0.5}(y) &= \begin{cases} 1 - y & \text{if } \textbf{min?} \\ y & \text{otherwise} \end{cases} \\
a &= \frac{2}{u_{\text{soft}} - l_{\text{soft}}}, \quad b = -\frac{u_{\text{soft}} + l_{\text{soft}}}{u_{\text{soft}} - l_{\text{soft}}}, \\
c &= \frac{1 + \exp(-a(u_{\text{hard}} - b)) + \exp(-a(l_{\text{hard}} - b)) + \exp(-a(u_{\text{hard}} + l_{\text{hard}} - 2b))}{\exp(-a(l_{\text{hard}} - b)) - \exp(-a(u_{\text{hard}} - b))}, \\
d &= \frac{c}{1 + \exp(-a(l_{\text{hard}} - b))}.
\end{aligned}$$

The inverse of $g_{\boldsymbol{\lambda}}^{-1}(y)$ is defined as

$$g_{\boldsymbol{\lambda}}^{-1}(y) = b - \log\left(\frac{c}{\mathrm{cr}_{0.5}(y) - d} - 1\right)/a \tag{4}$$

For normalizing observed log loss values, we utilize $\boldsymbol{\lambda} = (\text{True}, 0, \log(10), 0, \max\{\mathbf{y}_0^{train}\})$. Here $\mathbf{y}_0^{train}$ is the log loss value at the first epoch of the trajectories to train AutoML models. To normalize accuracy curves, we apply $\boldsymbol{\lambda} = (\text{False}, 0, 1, 0, 1)$.

LSTM. The LSTM implementation is adapted from the published code associated with NRI (Kipf et al. 2018). This implementation employs teacher forcing by utilizing the observation window. Specifically, it features a `step()` module, which comprises a Long-short-term memory (LSTM) block and a two-layer MLP with ReLU activation. The `step()` function processes the previous input state and hidden state, outputting the prediction and hidden state for the next immediate time step. The `forward()` function iteratively calls `step()` for a total of T_{max} times. During the initial T_{cond} steps, the observed learning curve data is used as the input state. For the subsequent steps beyond T_{cond}, the output from the previous prediction is used as the new input state.

Additional Results

Table 4: Extrapolation error for test loss curves derived from 2 tabular tasks and 2 image classification tasks computed over three prediction lengths, observing 10 epochs.

	car						segment					
	MAPE			RMSE			MAPE			RMSE		
Epochs	80	140	200	80	140	200	80	140	200	80	140	200
LC-BNN	0.4545	0.3715	0.3295	0.2798	0.2348	0.2096	0.4322	0.3893	0.3785	0.2358	0.2036	0.1877
LC-PFN	0.0723	0.0906	0.0999	0.0349	0.0397	0.0425	0.0795	**0.0890**	0.0937	**0.0329**	**0.0348**	**0.0361**
VRNN	0.2216	0.2054	0.2054	0.1306	0.1281	0.1281	0.3308	0.3660	0.3660	0.1518	0.1660	0.1660
LSTM	0.1104	0.1517	0.1733	0.0539	0.0697	0.0774	**0.0782**	0.0953	0.1076	0.0343	0.0386	0.0429
ODE	0.2110	0.2183	0.2218	0.0909	0.0927	0.0939	0.1665	0.1662	0.1670	0.0623	0.0635	0.0649
SDE	0.2039	0.2141	0.2183	0.0884	0.0900	0.0906	0.1493	0.1574	0.1607	0.0558	0.0581	0.0593
LC-GODE	**0.0644**	**0.0722**	**0.0765**	**0.0292**	**0.0307**	**0.0317**	0.0816	0.0892	**0.0925**	0.0342	0.0368	0.0378

	cifar10						cifar100					
	MAPE			RMSE			MAPE			RMSE		
Epochs	80	140	200	80	140	200	80	140	200	80	140	200
LC-BNN	0.2978	0.3847	0.5879	0.1990	0.1963	0.2174	0.2843	0.2157	0.1867	0.2656	0.2155	0.1871
LC-PFN	0.2274	0.3797	0.6008	0.1166	0.1519	0.1961	0.0557	0.0922	0.1505	0.0542	0.0834	0.1282
VRNN	0.2396	0.2386	0.2378	0.1301	0.1371	0.1407	0.2138	0.2131	0.2113	0.1652	0.1668	0.1661
LSTM	0.1851	0.1993	0.2093	0.1110	0.1092	0.1017	0.0502	0.0613	0.0709	0.0526	0.0614	0.0648
LatentODE	0.1655	0.1810	0.2039	0.1017	0.1026	0.0978	0.0705	0.0808	0.0823	0.0708	0.0788	0.0775
LatentSDE	0.1518	0.1555	0.1639	0.0956	**0.0913**	**0.0839**	0.0477	0.0597	0.0723	0.0492	0.0587	0.0647
LC-GODE	**0.1487**	**0.1536**	**0.1629**	**0.0953**	0.0926	0.0860	**0.0460**	**0.0571**	**0.0630**	**0.0481**	**0.0577**	**0.0598**

Table 5: Elapsed time to train each epoch and perform inference on the test set.

	car		segment		cifar10		cifar100	
	Train	Test	Train	Test	Train	Test	Train	Test
LC-BNN	0.0350	0.0183	0.0395	0.0558	0.0333	0.2833	0.0381	0.2814
LC-PFN	-	9.6855	-	9.4088	-	109.8517	-	112.8367
VRNN	0.4180	10.6489	0.4053	12.1125	1.7594	88.9615	1.5898	128.4061
LSTM	0.8364	0.0005	0.8794	0.0005	6.5582	0.0043	6.7457	0.0049
LatentODE	1.4698	0.1647	1.4684	0.1631	1.3543	0.1770	1.3206	0.1715
LatentSDE	1.5931	0.3403	1.5817	0.3342	1.3347	0.3229	1.4389	0.3494
LC-GODE	3.1136	0.8692	3.1476	0.8325	3.3043	0.7077	3.3879	0.7572

Table 4 shows the extrapolation error of six baselines and LC-GODE for test loss curves originated from 2 tabular tasks and 2 image classification tasks computed over three prediction lengths, observing 10 epochs. Table 5 shows the runtime (in seconds) of training one epoch and that of performing inference on the entire test set. For LC-PFN, the model is trained on synthetic curves drawn from a prior distribution and therefore no further training is needed, as long as the test learning curves are normalized according to the above description. Note that the inference time for Bayesian approaches, except for LC-BNN, is much greater than other approaches.

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*.

Kipf, T.; Fetaya, E.; Wang, K.-C.; Welling, M.; and Zemel, R. 2018. Neural relational inference for interacting systems. In *International conference on machine learning*, 2688–2697. PMLR.

Loshchilov, I.; and Hutter, F. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

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.

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.