Neural Network Nowcasting

1 Introduction

Modern deep learning models, such as large language models, are trained using classical adaptive optimizers like Adam [3] . These optimizers iteratively update neural network parameters, $\theta \in \mathbb{R}^n$, by applying gradient descent at each training step. The gradient descent process is computationally expensive, as it requires updating parameters θ through repeated optimization steps, and this cost grows as model sizes and data requirements increase, making the ability to predict and forecast parameters crucial for reducing computational overhead.

It has been shown that model parameters exhibit predictable trends over the course of training [2, 6]. Leveraging this predictability allows for *nowcasting*—the prediction of near-future parameters, enabling optimizers to skip certain steps and save computational resources. Nowcasting frameworks, such as the **Neuron Interaction and Nowcasting (NiNo) architecture** [4] , demonstrate significant improvements in training efficiency when used in conjunction with base optimizers like Adam.

In this paper, we demonstrate our recent improvements against the original NiNo model. In general, our contributions are four-fold (1). We propose and evaluate the use of a dynamic period scheduling strategy when applying NiNo. (2). We optimize the number of non-trival eigenvectors for Laplacian Positional Encoding for NiNo. (3) We incorporate eigenvector centrality as an additional node feature. (4) We enable and test NiNo's ability to use LSTM and GRU heads during training and inference.

2 Related Work

2.1 Parameter Prediction and Generation

Predicting future parameters to reduce training costs has been explored through models like IntrospectionMLP [6] and Weighted Neural Networks (WNNs) [2], which use MLPs to predict parameter updates based on past values. However, these methods predict parameters independently, missing interactions between parameters and assuming a fixed future horizon k, limiting their adaptability to different optimization stages. NiNo [4] builds on these approaches by leveraging neural network structure, conditioning predictions on k, and adapting to task-specific parameter trends, thereby enhancing efficiency.

2.2 Representation Learning of Neural Network Parameters

Representation learning in neural networks has focused on neuron permutation symmetry, where preserving the permutation of neurons across layers maintains network function [1]. Kofinas et al. [5] introduced *neural graphs*, which allow for Graph Neural Networks (GNNs) to effectively process the structure of neural networks. While effective, this approach doesn't properly model the neuron permutation symmetry of Transformers, but these shortcomings were addressed by [4]. In multi-head self attention, they propose restricting permutations across attention heads and relaxing permutations on weights $\mathbf{W}^q$ and $\mathbf{W}^k$, which was shown to result in better parameter prediction for transformers when coupled with their NiNo model.

3 Problem Formulation

Given a neural network $\mathcal{N}$, the order of the current epoch $T = t$, and the parameter $\boldsymbol{\theta}^i$ at $T = t$ denoted as $\boldsymbol{\theta}^i_t$, we aim to find a mapping $f^\phi(*)$ such that: $\hat{\boldsymbol{\theta}}^i_{t+k} = \boldsymbol{\theta}^i_t + f^\phi(\boldsymbol{\theta}^i_t, \boldsymbol{\theta}^i_{t-1}, \boldsymbol{\theta}^i_{t-2}, \dots)$. For example, if $j = k = 5$, to predict the parameters of the model at the 10th epoch, the parameters

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

required are those from the 5th, 4th, 3rd, 2nd, and 1st epochs during training. Such a mapping f^{ϕ} should generalize across diverse tasks.

4 Methodology

Terminology: In this section, FM-X refers to FashionMNIST with X channels per layer, and C10-Y refers to CIFAR-10 with Y channels per layer.

4.1 Dynamic Period

In the previous work, NiNo prediction steps are applied at a constant interval referred to as the period [4]. In this work, we observe that NiNo predictions are very effective early in the training process, then slow down the training process if applied too frequently as the model nears its target accuracy. To address this, we introduce the notion of a dynamic period, where the period gradually increases such that we apply NiNo very often during the early training steps, and very rarely as the model nears convergence. We conduct three experiments to illustrate the effectiveness of this method.

Number of Training Steps to Target Accuracy for Static and Dynamic Period: We conducted three trials for each of C10-16 and FM-16 (in-distribution), and C10-32, FM-32, and C100-32 (out-of-distribution), and measured the number of training steps required to reach convergence for both the static and dynamic period. We decided not to test the number of training steps reduced from ADAM given that this is already significantly proven for these tasks [4]. For this experiment, the static period is 1000, and the dynamic period varies by Equation 1 in the Appendix.

Training Time to Target Accuracy for Static and Dynamic Period: We conducted three trials for each of the models in the first experiment, but instead compared training time in seconds for the static and dynamic periods as the previous experiment. In this experiment, we also tested the training time to reach target accuracy for no Nino predictions (ADAM) to see how much time Nino saves when accounting the time spent constructing the GNN and the Nino prediction, as we feel such an analysis is missing in prior contributions.

Number of Training Steps to Target Accuracy for Short, Medium, Long, and Dynamic Period: To show that the reduction in number of training steps didn't come from solely more predictions rather than more optimally timed predictions. The period short, medium, and long lengths we tested were 100, 500, and 1000, respectively.

4.2 Neural Graph Construction with Enhanced Node Features

Edge Features: We adopt the representation of Neural Graph from the previous NiNo paper [4]. We represent all model parameters θ and auxiliary connections θ', such as residual and head connections, using edge features $\mathbf{E} \in \mathbb{Z}^{|\mathcal{V}|\times|\mathcal{V}|\times d_{\mathcal{E}}}$, ensuring that $||\mathbf{E}||_0 = |\theta| + |\theta'|$. To differentiate between θ and θ', an integer edge type $\mathbf{E}^{\text{type}} \in \mathbb{Z}^{|\mathcal{V}|\times|\mathcal{V}|\times 1}$ is assigned to each edge. Thus, our neural graph $\mathcal{G}(\theta) = (\mathcal{V}, \mathbf{E}^{\text{type}}, \mathbf{E})$ captures all relevant model parameters and connections, Figure 2

Node Features: For node features, we employ **Laplacian Positional Encoding (LPE)**, which helps GNNs capture structural information by embedding nodes of the same layer and type close to each other. This implicit proximity reflects layer-wise relationships that are essential for understanding complex model architectures. To compute LPE, we use **unweighted edges** and convert the graph to an **undirected** form. The 5 smallest non-trivial eigenvectors are extracted to serve as node features $\mathbf{V}^{\text{lpe}} \in \mathbb{R}^{|\mathcal{V}|\times 5}$. To determine the optimal number of eigenvectors for LPE, we performed an eigenvalue spectrum analysis of the graph Laplacian. Figure 1 shows the eigenvalue spectrum of the Laplacian for a representative neural graph (denoted as the C10 Laplacian).

Additionally, we employ **Eigenvector centrality**. A node with high eigenvector centrality is not only well-connected but is also connected to other nodes with high centrality. Formally, eigenvector centrality assigns a score $\mathbf{v}_i$ to each node i such that: $\mathbf{v}_i = \frac{1}{\lambda}\sum_{j\in\mathcal{N}(i)} A_{ij}\mathbf{v}_j$, where λ is the eigenvalue, A_{ij} is the adjacency matrix element, and $\mathcal{N}(i)$ represents the neighbors of node i.By incorporating eigenvector centrality as a node feature, we allow the model to better distinguish between the different nodes within the graph, such as those representing convolutional layer parameters or linear layer parameters. The eigenvector centrality feature is computed for each node and added to the node

feature representation. The updated node feature is defined as:

$$\mathbf{V}^0 = \phi_{\text{lpe}}(\mathbf{V}^{\text{lpe}}) + \phi_w(\mathbf{V}^w) \longrightarrow \mathbf{V}^{0*} = \phi_{\text{lpe}}(\mathbf{V}^{\text{lpe}^*}) + \phi_{\text{ec}}(\mathbf{V}^{\text{ec}}),$$

where $\mathbf{V}^{\text{ec}} \in \mathbb{R}^{|\mathcal{V}|\times 1}$ is the eigenvector centrality feature, ϕ_{lpe} and ϕ_{ec} are linear layers, and $\mathbf{V}^0 \in \mathbb{R}^{|\mathcal{V}|\times D}$ is the final node feature representation projected to the hidden dimension D.Since our testing is constrained to convolutional neural network (CNN) architectures, the positional feature for word embeddings ($\mathbf{V}^w$) is omitted in this work.

Integrated Neural Graph: We consider the history of c prior parameter vectors following the original NiNo[4] and WNNs[2]: $\Theta_{\tau:\tau_c} = [\theta_\tau, \theta_{\tau-1}, \ldots, \theta_{\tau_c}] \in \mathbb{R}^{n\times c}$, where $\tau_c = \tau - c + 1$. To let the multi layer perceptrons (MLP) DMS head capture temporal dependencies, we stack edge features over time as $\mathcal{E}_{\tau:\tau_c} = [\mathbf{E}_\tau, \mathbf{E}_{\tau-1}, \ldots, \mathbf{E}_{\tau_c}] \in \mathbb{R}^{|\mathbf{V}|\times|\mathbf{V}|\times d_{\mathbf{E}}\times c}$. At the same time, since the neural network architecture doesn't change, the node features and the edge features remain fixed. The entire $\Theta_{\tau:\tau_c}$ is transformed into a unified neural graph, expressed as $\mathcal{G}(\Theta_{\tau:\tau_c}) = (\mathbf{V}^{0^*}, \mathbf{E}^{\text{type}}, \mathcal{E}_{\tau:\tau_c})$.

4.3 Direct multi-step forecasting (DMS)

During the training, the edge features are flattened into a vector, and then fed into the DMS head. The original NiNo adopted MLP for the DMS prediction. Though the method is effective due to the pre-stacking of the parameters over time, the inherent limitation of MLP in time series forecasting inspires us to try out other neural network architecture for the final prediction. In this work, we propose Gated Recurrent Units (GRUs) and Long Short-Term Memory (LSTM) networks as replacements for the simpler MLP for the DMS head. These recurrent architectures maintain a memory of past inputs and adaptively update it over time, which could theoretically improve the model performance.

5 Dataset

NiNo's vision task adaptation was trained on a meta-dataset of 600 models from FashionMNIST and CIFAR-10, using two convolutional architectures with 16-32-32 and 32-64-64 channels. Each architecture and dataset contributed 300 models, capturing 1.56 million checkpoints across training stages. This dataset allowed NiNo to learn generalized patterns from diverse model states, enhancing its ability to perform well across various image classification tasks.

The datasets used for our experiments include in-distribution (ID) and out-of-distribution (OOD) image classification datasets. The in-distribution datasets were FM-16 and C10-32, and the out of distribution tasks were FM-32, C10-32 and C100-32.

6 Results and Discussion

6.1 Dynamic Period

Number of Training Steps to Target Accuracy for Static and Dynamic Period: The results from this experiment can be found in Table 2 in the appendices. From this table, we can see that a dynamic period outperforms a static period for all 5 tasks with respect to number of training steps. We find this result to be particularly significant as three of the five tasks are out-of-distribution.

Training Time to Target Accuracy for Static and Dynamic Period: The results from this experiment can be found in Table 3 in the appendices. From this table, we can see that a dynamic period outperforms a static period for all 5 tasks with respect to training time. We find this result to be particularly significant as three of the five tasks are out-of-distribution. We can also see that even when we include the time spent applying NiNo to the training process, both a static and a dynamic period decrease the total training time.

Number of Training Steps to Target Accuracy for Short, Medium, Long, and Dynamic Period: The results from this experiment can be seen in Figure 4. In this experiment, a dynamic period outperforms all of the relevant static period lengths.

6.2 Enhancing Node Representations

Eigenvalue Spectrum Analysis Figure 1 shows the eigenvalue spectrum of the Laplacian for a representative neural graph (denoted as the C10 Laplacian). The plot exhibits a characteristic "elbow" pattern, where the eigenvalues increase sharply before stabilizing. This behavior is indicative of the transition between the smallest non-trivial eigenvectors and the higher-frequency eigenvectors that may capture noise rather than meaningful structure. This choice ensures that the LPE effectively encodes the positional relationships of nodes in the graph, enabling the model to better leverage structural information.

Number of Training Steps for Enhanced Node Representation The results from this experiment can be found in Table 4. Across both in-distribution and out-of-distribution tasks, we see a reduction in the number of steps needed to reach the target accuracy with respect to each task.The number of steps recorded for NiNo ($\mathbf{V}^0$) and NiNo ($\mathbf{V}^{0*}$) was the median number of steps taken after 5 training runs for each task. Approximately 200 optimization steps were reduced for the C10-16 and FM-16 tasks and a slight reduction was seen for the C10-32 task when comparing between NiNo ($\mathbf{V}^0$) and NiNo ($\mathbf{V}^{0*}$). Clearly, the enhanced node representation $\mathbf{V}^{0*}$ effectively combines positional and centrality information, providing a richer and more informative feature space for the NiNo architecture to operate on. This improvement enables the model to better capture the structural nuances of the neural graph, leading to the improved optimization performance we observed.

6.3 Direct Multi-Step Forecasting Architecture

Note: In this subsection, all the experiments are conducted for OOD task C10-32.

Loss Trajectory for Different DMS Heads and Prediction Horizons Fig. 3 and 4 show the loss of using different DMS heads and using the LSTM head, $ctx = 10$ but different hor value when training the NiNo. As we can infer from the two graphs, both GRU and LSTM outperform the MLP head from the perspective of loss when $ctx = 10$ and $hor = 40$, and when the ctx is fixed, the longer the hor is, the smaller the loss becomes.

Number of Training Steps and Time Using Different DMS Heads As shown in Fig. 5, when setting $ctx = 10$ and $hor = 200$, all three heads reduced the number of steps compared to the Adam. The number of steps and time it takes to reach the target accuracy is displayed in the Table. 1. However, as shown in the table, fewer training steps do not always guarantee shorter training time. The GRU head here, though taking less steps, the execution time is longer compared to Adam. This is due to the overhead of computing graph features for the network during NiNo steps. Yet in general, NiNo does speed up the training process.

Number of Training Steps and Time Using LSTM head and Different Horizons Figure 8 and Table 2 show that larger prediction horizons generally speed up training, but not always linearly (e.g., $hor = 400$ outperforms $hor = 500$ and $hor = 600$). Another phenomenon we observed is that LSTM head with $hor = 300$ failed after 7000 steps. Such failure also exists in other heads, as demonstrated in Fig. 7. This is because incorrect parameter predictions might trap the model in gradient regions which is impossible to escape. Such failures highlight the need for future protection mechanisms to roll back the model when failures occur.

7 Conclusion and Future Work

In this work, we presented significant advancements to the NiNo architecture, improving its efficiency in training neural networks by leveraging parameter predictability. Our contributions include a dynamic period scheduling strategy, optimized node representations through Laplacian Positional Encoding (LPE) and eigenvector centrality, and the integration of advanced forecasting architectures such as GRUs and LSTMs. Experimental results demonstrate that these enhancements reduce training steps and time across both in-distribution and out-of-distribution tasks, highlighting the efficacy of these improvements upon the existing NiNo architecture.

Future work includes unifying contributions into a NiNo++ framework, adding a rollback mechanism to address failed nowcasts, and testing on larger, more costly models. Overall, these extensions aim to enhance the applicability and effectiveness of NiNo in large-scale machine learning tasks.

References

[1] Robert Hecht-Nielsen. On the algebraic structure of feedforward network weight spaces. pages 129–135, 1990.

[2] Jinhyeok Jang, Woo-han Yun, Won Hwa Kim, Youngwoo Yoon, Jaehong Kim, Jaeyeon Lee, and ByungOk Han. Learning to boost training by periodic nowcasting near future weights. *PMLR*, pages 14730–14757, 2023.

[3] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[4] Boris Knyazev, Abhinav Moudgil, Guillaume Lajoie, Eugene Belilovsky, and Simon Lacoste-Julien. Accelerating training with neuron interaction and nowcasting networks. *arXiv preprint arXiv:2409.04434*, 2024.

[5] Miltiadis Kofinas, Boris Knyazev, Yan Zhang, Yunlu Chen, Gertjan J Burghouts, Efstratios Gavves, Cees GM Snoek, and David W Zhang. Graph neural networks for learning equivariant representations of neural networks. *arXiv preprint arXiv:2403.12143*, 2024.

[6] Abhishek Sinha, Mausoom Sarkar, Aahitagni Mukherjee, and Balaji Krishnamurthy. Introspection: Accelerating neural network training by learning weight evolution. *arXiv preprint arXiv:1704.04959*, 2017.

8 Appendix

8.1 Figures

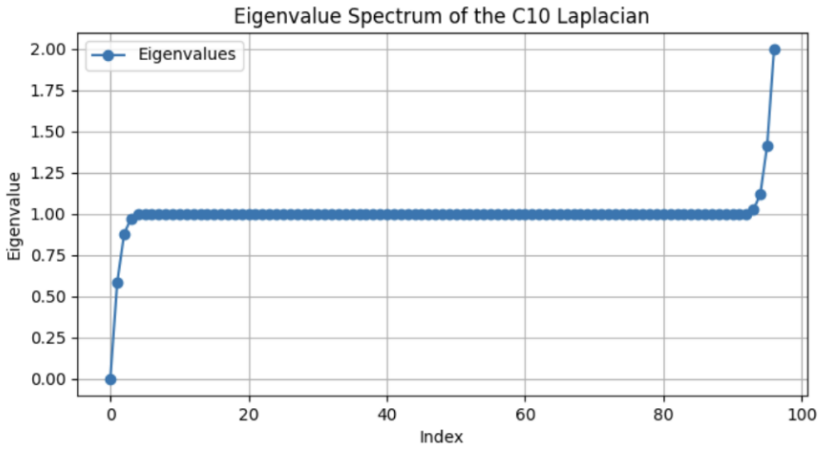

Figure 1: Eigenvalue Spectrum of the C10 Laplacian. The elbow point, corresponding to the transition from significant eigenvectors to noise, is marked around the fifth eigenvalue.

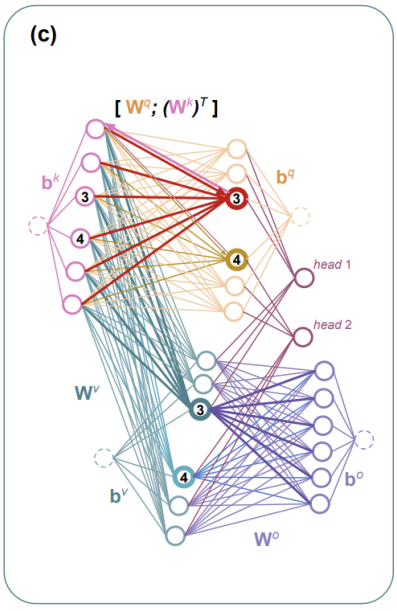

Figure 2: Neural graph with color-coded edge/parameter types

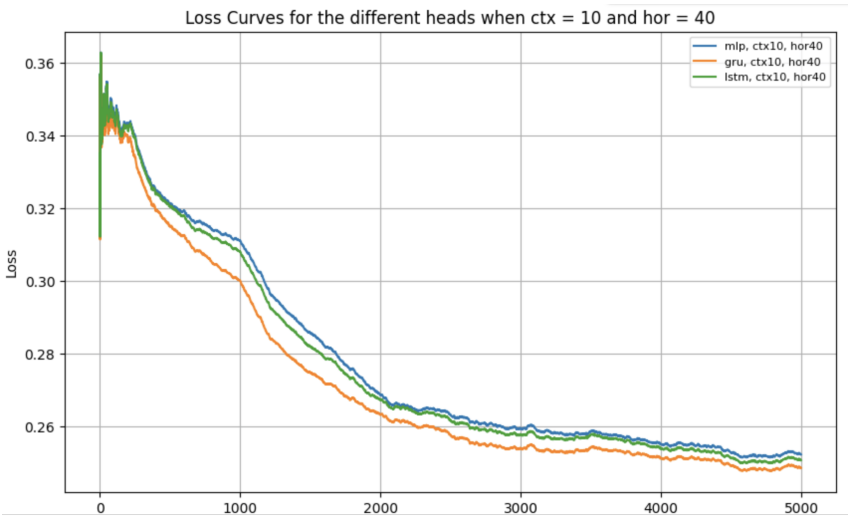

Figure 3: Training loss of NiNo with different DMS heads

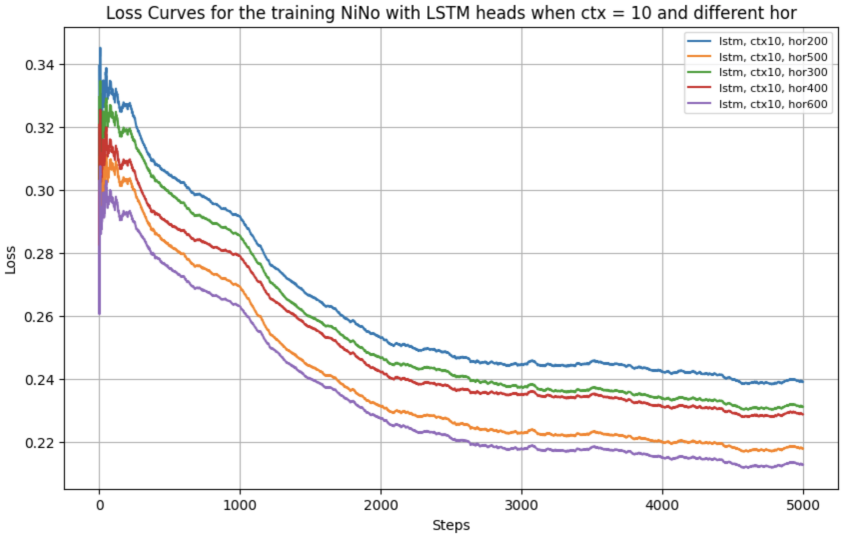

Figure 4: Training loss of NiNo with LSTM DMS head with different hor, ctx = 10

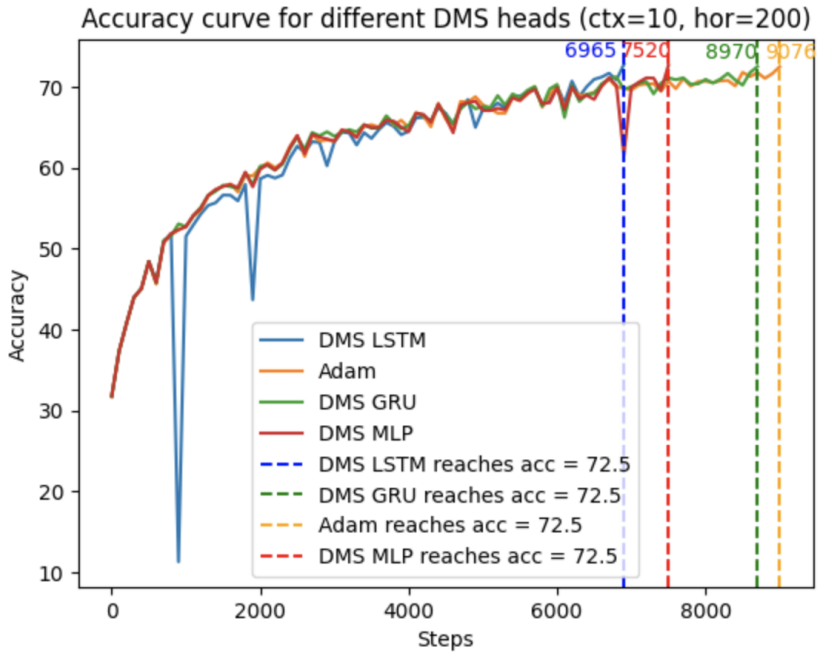

Figure 5: Accuracy curve when applying NiNo with different DMS heads, ctx and hor set to be 10 and 200

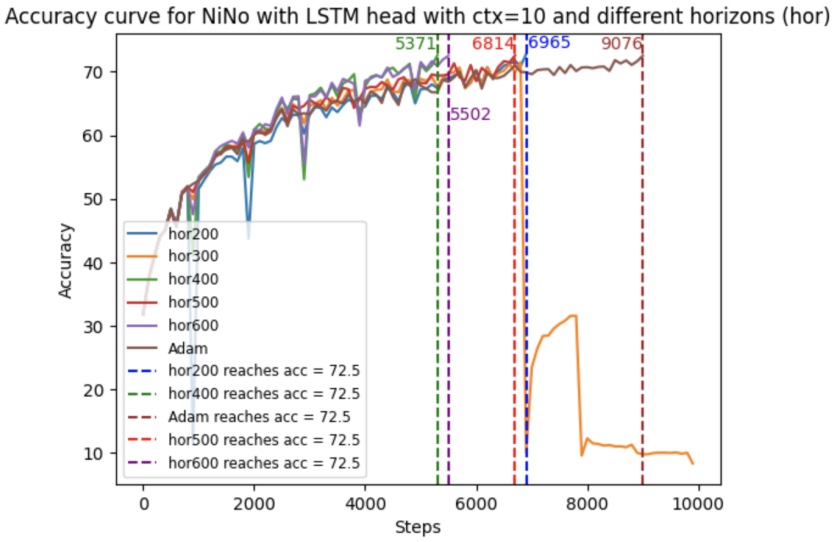

Figure 6: Accuracy curve when applying NiNo with LSTM DMS head with different hor, ctx = 10

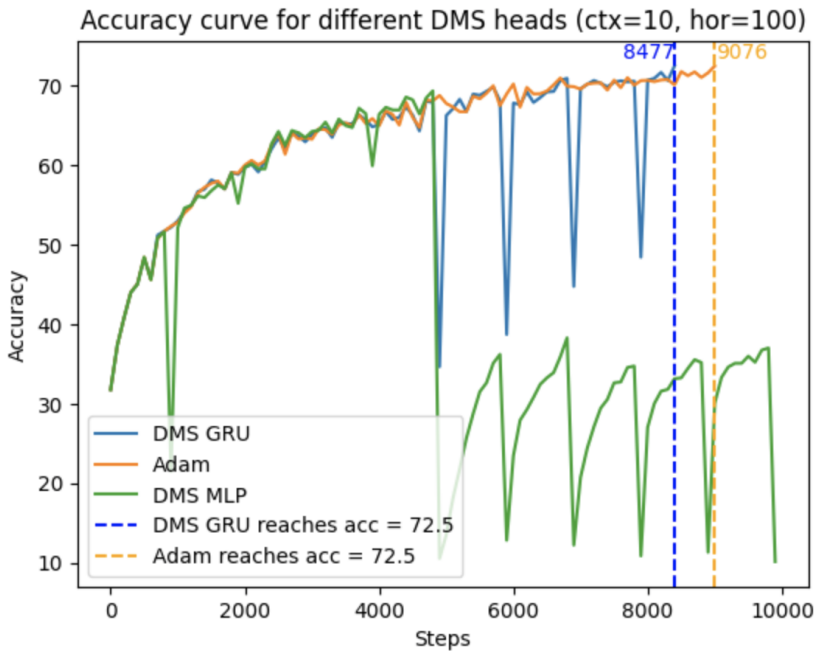

Figure 7: A failure case for MLP head

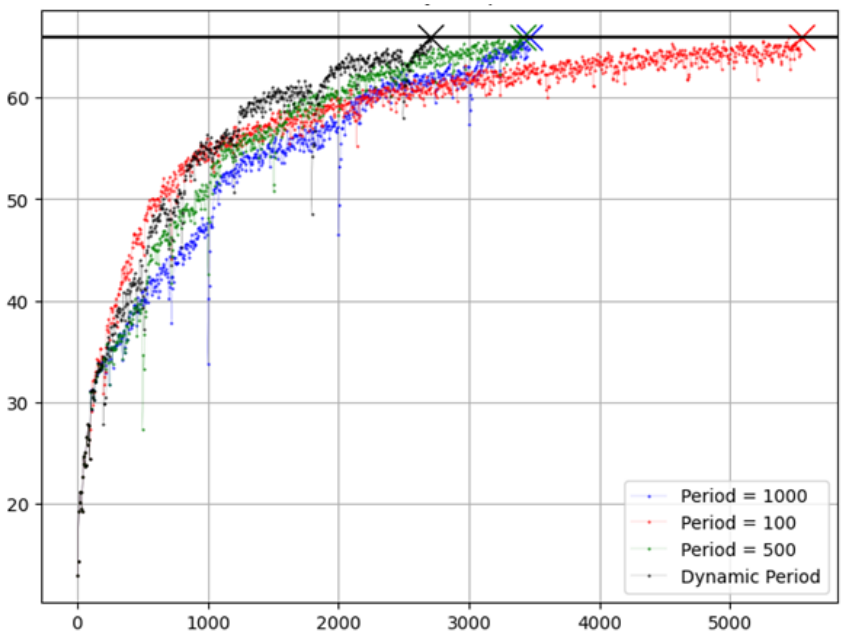

Figure 8: Accuracy vs Number of Training Steps for Different Period Sizes

8.2 Tables

Table 1: Number of Steps and Time to Reach 72.5% Accuracy for Different DMS Heads

DMS Heads/Adam	Steps to Reach 72.5% Accuracy	Time (in seconds)	Reduction in Time
LSTM	6965	113.73	30.00%
GRU	8970	163.54	-0.01%
MLP	7520	127.09	21.76%
Adam	9076	162.44	

Table 2: Number of Steps and Time to Reach 72.5% Accuracy for LSTM head and different Horizons, ctx = 10

Prediction Horizon	Steps to Reach 72.5% Accuracy	Time (in seconds)	Reduction in Time
200	6965	125.94	36.50%
400	5371	90.14	44.51%
500	6814	114.79	29.33%
600	5502	92.87	42.8%
Adam	9076	162.44	

Table 3: Number of Training Steps to Target Accuracy for Static and Dynamic Period (3 trials)

	FM-16	C10-16	FM-32	C10-32	C100-32
Static	5373	4129	4692	4820	4266
Dynamic	3755	2584	3553	3973	3500

Table 4: Training Time (seconds) to Target Accuracy for Static and Dynamic Period (3 trials)

	FM-16	C10-16	FM-32	C10-32	C100-32
ADAM	244	343	585	888	672
Static	197	195	420	489	433
Dynamic	143	122	319	403	266

Table 5: Number of Steps (Time (s)) Compared using $\mathbf{V}^0$ vs $\mathbf{V}^{0*}$

	In-Distribution Tasks	Out-of-Distribution Tasks	
	C10-16	FM-16	C10-32
NiNo ($\mathbf{V}^0$) steps	3717 (184)	4399 (162)	4966 (506)
NiNo ($\mathbf{V}^{0*}$) steps	3575 (178)	4155 (152)	4914 (501)