


Less is More: Efficient Weight “Farcasting” with 1-Layer Neural Network

Xiao Shou ¹, Debarun Bhattacharjya², Yanna Ding³, Chen Zhao¹, Rui Li⁴,
and Jianxi Gao³

¹ Baylor University, Waco, TX 76706, USA
{Xiao_Shou, Chen_Zhao}@baylor.edu

² IBM AI Research, Yorktown Heights, NY 10598
{debarunb@us.ibm.com}

³ Rensselaer Polytechnic Institute, Troy, NY 12180
{dingy6,gaoj8}@rpi.edu

⁴ Amazon
{ruilit@amazon.com}

Abstract. Addressing the computational challenges inherent in training large-scale deep neural networks remains a critical endeavor in contemporary machine learning research. While previous efforts have focused on enhancing training efficiency through techniques such as gradient descent with momentum, learning rate scheduling, and weight regularization, the demand for further innovation continues to burgeon as model sizes keep expanding. In this study, we introduce a novel framework which diverges from conventional approaches by leveraging long-term time series forecasting techniques. Our method capitalizes solely on initial and final weight values, offering a streamlined alternative for complex model architectures. We also introduce a novel regularizer that is tailored to enhance the forecasting performance of our approach. Empirical evaluations conducted on synthetic weight sequences and real-world deep learning architectures, including the prominent large language model DistilBERT, demonstrate the superiority of our method in terms of forecasting accuracy and computational efficiency. Notably, our framework showcases improved performance while requiring minimal additional computational overhead, thus presenting a promising avenue for accelerating the training process across diverse tasks and architectures.

Keywords: Neural Network Training · Long-term Forecasting · (Stochastic) Gradient Descent Optimization · Language Model · Deep Learning Efficiency.

1 Introduction

Large-scale deep learning systems such as large language models (LLMs) typically require significant investments of time and computation for training. For instance, GPT-3 [2] demands several days of training even on high-performance machines, posing substantial challenges in terms of efficiency and resource allocation. Over

the years, numerous efforts have been dedicated to addressing the computational demands of training large-scale models [24]. Techniques such as architecture minimization and compression [30] and the deployment of efficient physical hardware systems have yielded improvements in efficiency [15].

From an algorithmic perspective, diverse training strategies aim to train the model in various ways, leading to fewer prediction errors, reduced data requirements, faster convergence, and other benefits. This enhanced quality can then be leveraged to create a smaller, more efficient model by trimming the number of parameters if needed. Two notable examples of optimization techniques include momentum-based methods like Adam and knowledge distillation-based methods like DistilBERT [27, 10]. While these methods contribute to learning stability and marginally accelerate the training process, they do not directly address the fundamental issue of reducing overall training time.

Efforts to cut down training time typically leverage short-term forecasting for predicting parameter weights; notable examples include *Introspection* [28] and the *Weight Nowcaster Network (WNN)* [13]. For instance, WNN incorporates two feed-forward neural networks for capturing both weight parameters and temporal differences to predict the weights of the target neural network over the next 5 steps, a technique termed “nowcasting”. Such approaches have several shortcomings: (i) they can only forecast in the short term, which limits their applicability to further reduce training time; (ii) they have only been applied to deep learning systems with fewer number of parameters – in particular, WNN involves complex transformations that inhibit their application in LLMs as demonstrated later in our experiment on DistilBERT [27]; (iii) they can only forecast to 1 future time step which may not be optimal if abrupt changes occur in the training.

We address limitation (i) by focusing on long-term weight prediction, which we term “farcasting”. This concept is akin to long-term time series forecasting, where we predict many steps ahead. The motivation for adopting time series forecasting techniques in predicting neural network weights over time is driven by several key factors. Firstly, numerous state-of-the-art techniques have been developed for long-term time series forecasting [35]. The training process of neural networks, where weights and their updates can be seen as multivariate vectors or multivariate observations, shares similarities with time series data. However, a major distinction exists: neural network weight updates are governed by specific update rules, while time series data typically exhibit periodicity and trends. To address limitation (ii), we explore the feasibility of using a simple model — a one-layer feed-forward neural network — for this forecasting task, which further reduces training time, marking a major leap forward in the efficiency of deep learning practices. By allowing our neural network to predict a sequence of future steps leveraging direct multi-step forecasting strategies [4], we naturally address limitation (iii) and enhance model stability by smoothing out abrupt parameter changes.

Our **contributions** are summarized as follows:

- We introduce the concept of "farcasting" for parameter prediction in the training of machine learning (ML) and deep learning (DL) systems, emphasizing large-scale models.
- We demonstrate that even a one-layer neural network can effectively solve linear systems, highlighting the potential for simplicity in forecasting.
- We show that a much smaller feed-forward fully connected network can significantly reduce the number of parameters while achieving accuracies that are equivalent or superior to those from larger networks.
- We present compelling applications in computer vision and LLMs, utilizing two notable architectures: convolutional neural networks (CNNs) and DistilBERT.

2 Related Work

Parameter Prediction of Deep Learning. A closely related line of work to ours focuses on accelerating deep learning training. A notable prior work, the Introspection Network [28], uses the weight history of unseen neural networks and sample points from these weights to predict future values using another introspection network, thereby speeding up the training process of the unseen neural network. Similarly, Knyazev et al. [18] expanded on Graph HyperNetworks [36] by training a hypernetwork to predict a set of performant parameters for the source task. However, since these parameters are often suboptimal, they serve as initialization for further training. In contrast, our approach diverges by forecasting the trajectory of weights rather than predicting a single point value for the neural network. This strategy largely circumvents the need for further training. Another relevant work is the Weight Nowcaster Network (WNN), which periodically nowcasts near-future weights [13]. Unlike these methods, which only predict a few steps ahead, our approach tackles long-term forecasting over many steps, significantly boosting the training of large-scale deep learning systems.

Time Series Forecasting. Our work is closely related to long-term time series forecasting. Various transformer-based models, such as [32, 25], offer state-of-the-art predictive performance. However, the computational demands of transformers may be prohibitive in our setting. Alternatively, foundation model approaches have been explored, as noted by Ye et al. [33]. In our context, feed-forward neural networks present a more feasible option. Previous relevant work includes DLinear and its variants by Zeng et al. [35], as well as methods incorporating transformations to capture periodicity and trends, as discussed in a more recent work [19]. However, periodicity may be irrelevant in our setting since weight sequences do not exhibit periodic properties.

ODE Forecasting. Numerous methods have been proposed for time series forecasting, utilizing convolutional networks, neural Ordinary Differential Equations (ODEs) [3], and transformers. Temporal convolutional neural networks [22] hierarchically aggregate adjacent timestamps to capture time series patterns. However, these approaches often rely on previously predicted data for future value rollouts,

leading to inaccuracies in long-term forecasting due to error propagation. Neural ODEs fit observed data to a system of differential equations, while Graph ODEs [34, 11, 12, 23, 5] extend this concept to model coupled dynamical systems. Nevertheless, training differential equation models involves computationally intensive numerical integration, especially for large networks. In this study, we investigate discrete time forecasting approaches to address the modeling of parameter trajectories.

Large Scale Optimization. Modern large-scale optimization and deep learning primarily utilize first-order methods, such as mini-batch stochastic gradient descent and its momentum-based variants like Adam [17] and AdamW [21], to achieve faster convergence. Additional techniques include learning rate scheduling [9, 20], applying regularizers to trainable parameters in networks [26, 14], and various initialization strategies [7, 8]. Some researchers have also explored optimization algorithms by framing it as learning to learn an optimizer [1]. Our approach is orthogonal to these efforts; we instead frame the problem as a forecasting task.

3 Linear Regression: An Example

In large-scale machine and deep learning endeavors, the primary goal is to minimize a designated loss function L concerning certain parameters ϕ within a defined parameter space. A prevalent optimization technique in deep learning is mini-batch stochastic gradient descent (SGD), where a sequence of weights and biases in ϕ and represented as $(\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_n)$ (where \mathbf{w}_i includes both weights and biases) undergoes n iterations of weight updates: $\mathbf{w}_{i+1} \leftarrow \mathbf{w}_i - \alpha \nabla L(\mathbf{w}_i)$.

For instance, consider a typical problem in machine and deep learning, such as linear regression, a common statistical and machine learning task. The objective is to minimize the sum of squared errors. Given a data matrix $\mathbf{D} \in \mathbb{R}^{n \times d}$ comprising n samples, each with d features, and a response vector $\mathbf{e} \in \mathbb{R}^n$ containing n responses, the aim is to find a weight vector $\mathbf{w} \in \mathbb{R}^d$ that minimizes the squared loss between predicted responses and the ground truth:

$$\min_{\mathbf{w}} \frac{1}{2} \|\mathbf{D}\mathbf{w} - \mathbf{e}\|_2^2 \quad (1)$$

This represents an unconstrained quadratic problem with a unique optimal solution $\mathbf{w}^* = (\mathbf{D}^\top \mathbf{D})^{-1} \mathbf{D}^\top \mathbf{e}$. Gradient descent, a common optimization method, iteratively generates a sequence of \mathbf{w}_i 's using updates of the form $\mathbf{w}_{i+1} \leftarrow \mathbf{w}_i - \alpha \nabla L(\mathbf{w}_i)$, where $\nabla L(\mathbf{w}_i) = (\mathbf{D}^\top \mathbf{D})\mathbf{w}_i - \mathbf{D}^\top \mathbf{e}$ for the quadratic optimization problem.

In contrast, a feed forward neural network $g_\theta(\cdot)$ offers a more expressive modeling approach, aiming to minimize the sum of squared errors between the predicted and ground truth parametrized by weights and bias in a neural network:

$$\min_{\theta} \frac{1}{n} \|g_\theta(\mathbf{D}) - \mathbf{e}\|_2^2 \quad (2)$$

Here, $g_\theta(\mathbf{D}) = \sigma_k(\dots\sigma_2(\sigma_1(\mathbf{D} \cdot \mathbf{W}_1 + \mathbf{b}_1) \cdot \mathbf{W}_2 + \mathbf{b}_2\dots)$ represents a k -layer neural network with weights \mathbf{W}_i and biases \mathbf{b}_i , and activation functions σ_i such as RELU. For the above minimization problem, the final activation is identity. Notably, this problem is generally non-convex, lacking a guaranteed unique optimal solution. Mini-batch stochastic gradient descent optimizer or one of its variants is typically employed for training the network. We will consider the two afore-mentioned cases of regression in the following sections to illustrate our approach through synthetic data experiments.

4 Method

Analogous to long-term time series forecasting (LTSF), we are given a sequence of weights from (stochastic) gradient descent or one of its variants of $n + 1$ steps, denoted as $\mathbf{X} = (\mathbf{w}_0, \mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_n)$ and our goal is to predict future weights m steps ahead, denoted as $\mathbf{Y} = (\mathbf{w}_{n+1}, \mathbf{w}_{n+2}, \dots, \mathbf{w}_{n+m})$, where $m \gg 10$. For simplification, our prediction can be expressed as $\mathbf{X} \in \mathbb{R}^{d \times (n+1)} \rightarrow \mathbf{Y} \in \mathbb{R}^{d \times m}$. Figure 1 illustrates our problem using 2 dimensional linear regression.

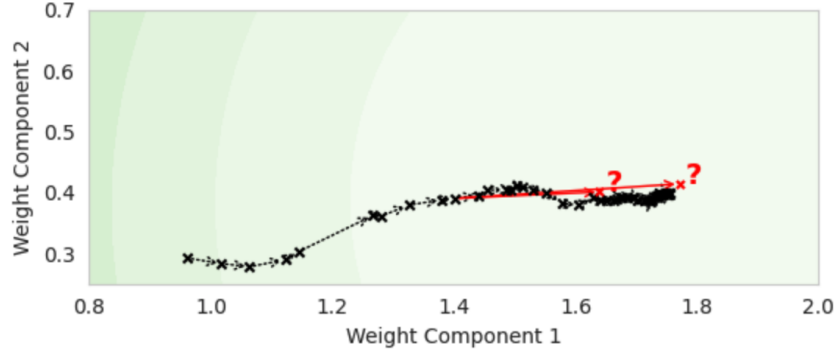


Fig. 1: Illustration of the weight forecasting procedure. Weight sequences are generated for a two-dimensional linear regression problem from Equation 1. Black points are weights from SGD, red points are predictions from forecasting. The red arrow \rightarrow shows farcasting from a partial weight sequence.

We pursue the direction of direct multi-step forecasting [35] instead of iterated multi-step (IMS) [29] due to more error accumulation from each autoregressive step in IMS. We emphasize that here (\mathbf{X}, \mathbf{Y}) specifically denotes the input output pairs which are parameter (sub)sequences from training or optimization processes. The ultimate objective is to significantly reduce optimization steps, crucial for large-scale machine and deep learning systems. While there are many state-of-the-art transformer based models for LTSF such as [25], they are not suitable here as the computational overhead for training such large models may well

be greater than performing stochastic gradient update for the current system. We thus restrict our attention to feed forward neural networks. In particular, we are interested in finding a neural network $h \in \mathcal{H}$ in the space of all neural networks that maps from a sequence of weights $\mathbf{X} \in \mathbb{R}^{d \times (n+1)}$ from (stochastic) gradient descent or its variants to a $\mathbf{Y} \in \mathbb{R}^{d \times m}$. For a simple 1 layer neural network $h_\theta : \mathbf{X} \rightarrow \mathbf{Y}$ where parameter set θ contains weights and bias $\{\mathbf{A}, \mathbf{b}\}$, we aim to find a solution for the system of linear equations: $\mathbf{X} \cdot \mathbf{A} + \mathbf{b} = \mathbf{Y}$. In essence, our method learns a mapping from past weight sequences to future weights using a simple feedforward layer, enabling multi-step forecasting and reducing the need for backpropagation at every step. For a simplified system, we arrive at the following statement.

Proposition 1. *If the (stochastic) gradient descent (or its variant) update (i.e., $\mathbf{w}_{i+1} \leftarrow \mathbf{w}_i - \alpha \nabla L(\mathbf{w}_i)$) can be written as $\mathbf{w}_{i+1} = c_i \cdot \mathbf{w}_i + d_i$ for some $c_i, d_i \in \mathbb{R}$, then there exists \mathbf{A} and \mathbf{b} that solve $\mathbf{X} \cdot \mathbf{A} + \mathbf{b} = \mathbf{Y}$.*

Proof sketch. For this linear system of equations, we have $[\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_n] \cdot \mathbf{A}_i + \mathbf{b}_i = \mathbf{w}_{n+i}$ for $i \in \{1, 2, \dots, m\}$, or more compactly $\sum_{j=1}^n \mathbf{w}_j \cdot \mathbf{A}_{ji} + \mathbf{b}_i = \mathbf{w}_{n+i}$. We solve for the case where $i = 1$, and then $i = 2$ and so on. $\sum_{j=1}^n \mathbf{w}_j \cdot \mathbf{A}_{j1} + \mathbf{b}_1 = \mathbf{w}_{n+1}$. Note we also have $c_n \cdot \mathbf{w}_n + d_n = \mathbf{w}_{n+1}$. We can choose \mathbf{A}_1 to be some scalar multiplied by a one-hot vector with 1 on the last term, i.e. $\mathbf{A}_1 = c_n \cdot [0, 0, 0, \dots, 1]^\top$ and $\mathbf{b}_1 = d_n$. Similarly we can solve for the $i = 2$ case by noting $\mathbf{w}_{n+2} = c_n c_{n+1} \cdot \mathbf{w}_n + c_{n+1} d_n + d_{n+1}$ using our recurrence relation. Hence $\mathbf{A}_2 = c_n c_{n+1} \cdot [0, 0, 0, \dots, 1]^\top$ and $\mathbf{b}_2 = c_{n+1} d_n + d_{n+1}$. We can perform this procedure until m and then acquire \mathbf{A}^* to be a matrix such that the only row that contains nonzero entries - the n^{th} row - contains scalar multiples and the i^{th} entry is $c_n \cdot c_{n+1} \cdot \dots c_{n+i-1}$ for $i \in \{1, 2, 3, \dots, n\}$. \square

While the assumption that the update is linear is a simplification in modern large scale deep learning systems, it serves as a linear approximation to many complex systems. More significantly, a simple solution \mathbf{A}^* places the nonzero entries on the n^{th} row, which inspires our approach to design more efficient forecasting techniques. In practice, when we have a dataset with l pairs of \mathbf{X} and \mathbf{Y} , we seek a computation-friendly numerical optimization approach using neural networks to minimize the prediction error:

$$\min_{\{\mathbf{A}, \mathbf{b}\}} \frac{1}{l} \sum_{i=1}^l \|\mathbf{X}^i \cdot \mathbf{A} + \mathbf{b} - \mathbf{Y}^i\|_{l_1} \quad (3)$$

The entry-wise l_1 -norm is applied here due to the parameters from (stochastic) gradient descent or its are common larger at early steps are usually smaller at late ones and the prediction error should not be dominated by early ones. Another more involved approach is to employ some sort of preprocessing P or transformation ψ [19] on \mathbf{X} . We use preprocessing to indicate such process is non-trainable, and transformation for trainable approaches. They can be expressed as $\min_{\{\psi, \mathbf{A}, \mathbf{b}\}} \frac{1}{l} \sum_{i=1}^l \|\psi(P(\mathbf{X}^i)) \cdot \mathbf{A} + \mathbf{b} - \mathbf{Y}^i\|_{l_1}$. While this approach may working reasonably well in time series forecasting [19], the sequential nature of parameter

updates via (stochastic) gradient descent does not imply any composition of periodicity and trends commonly observed in time series data. Furthermore, such processing and transformation increases computational cost and leads to challenges in training the network as illustrated later by our experiments on DistilBERT.

Sampling Weights for Efficiency. A potential approach to further reduce computation is to use a subset of parameter vectors by constraining the previous linear objective function:

$$\begin{aligned} \min_{\{\mathbf{A}, \mathbf{b}\}} \quad & \frac{1}{l} \sum_{i=1}^l \|\mathbf{X}_T^i \cdot \mathbf{A} + \mathbf{b} - \mathbf{Y}^i\|_{l_1} \\ \text{s.t.} \quad & T \subset \{0, 1, 2, 3, \dots, n\} \end{aligned} \quad (4)$$

where \mathbf{X}_T^i is the (sampled) subset of columns from \mathbf{X}^i . If we restrict the cardinality to $|T| = k$, we can achieve this by drawing k columns. Much research has been conducted in this area, known as the column subset selection problem [31], which often involves column importance sampling algorithms such as LinearSVD and score sampling. However, in our setting, performing such decomposition procedures for large d (e.g., 10^8) is costly. A simple approach is to draw k columns without replacement from a uniform distribution over $(0, n)$ as a preprocessing step.

Importance of Weights for Learning and Prediction. Upon further reflection on the update rules commonly used in relevant optimizers, we observe that the most critical weight for predicting the weight at step $n + 1$ is the weight \mathbf{w}_n from step n , followed by the weight \mathbf{w}_{n-1} from step $n - 1$, and so on. This reliance on the most recent weight is a hallmark of first-order gradient-based optimizers commonly used in deep learning, such as SGD and Adam [17]. Additionally, we note that the solution \mathbf{A}^* in Theorem 1 assigns nonzero values only to the last term, further validating our proposed approach. Our method is straightforward and ‘deterministic’ — it selects the most recent weight \mathbf{w}_n , which is most relevant for future updates in our forecasting problem. Furthermore, we include the initial weight, as it determines the starting point and has been crucial in the study of weight initialization in deep learning [7, 8]. This importance is also reflected in our empirical investigations, demonstrating the significance of initial and final weights. Our approach offers the side benefits of reducing memory usage, runtime computation, and the number of parameters from the original $\mathcal{O}(nd)$ in Equation 3 to $\mathcal{O}(kd)$ in Equation 4, and finally to $\mathcal{O}(d)$ in Equation 5 below.

$$\min_{\{\mathbf{A}, \mathbf{b}\}} L_{pred}(\{\mathbf{A}, \mathbf{b}\}) = \frac{1}{l} \sum_{i=1}^l \|\mathbf{X}_{[0,n]}^i \cdot \mathbf{A} + \mathbf{b} - \mathbf{Y}^i\|_{l_1} \quad (5)$$

Background Knowledge from First Order Optimality Condition. The training process for most DL systems is largely non-convex, where a typical model converges to a (local) optimal. This naturally leads to the following assumption.

Assumption 1 According to the first order optimality condition for unconstrained optimization problems, the sequence of weights updated via (stochastic) gradient descent (or its variant) follows $\lim_{i \rightarrow \infty} \|\nabla L(\mathbf{w}_i)\|_{l_p} = 0$ with respect to some norm $1 \leq l_p \leq \infty$.

From Assumption 1, we can deduce $\|\nabla L(\mathbf{w}_n)\|_{l_1} \leq \|\nabla L(\mathbf{w}_0)\|_{l_1}$ for some large n . Meanwhile we have $\mathbf{w}_{i+1} - \mathbf{w}_i = -\alpha_i \nabla L(\mathbf{w}_i)$. Commonly in ML/DL, the learning rate remains constant or decays according to some annealing procedure such that $0 \leq \alpha_j \leq \alpha_0$ for $j \in \{1, 2, 3, \dots, n\}$. We arrive at $\|\mathbf{w}_1 - \mathbf{w}_0\|_{l_1} \geq \|\mathbf{w}_{n+1} - \mathbf{w}_n\|_{l_1}$ for some large n . Hence we impose a soft penalty L_{grad} to incorporate such knowledge for any deviation from this prediction:

$$L_{grad} = \sum_j^J \max(\|\hat{\mathbf{w}}_{j+1} - \hat{\mathbf{w}}_j\|_{l_1} - \|\mathbf{w}_1 - \mathbf{w}_0\|_{l_1}, 0) \quad (6)$$

for some large j and $\hat{\mathbf{w}}_j$'s are predicted weights from Equation 5. In practice, we can choose j to be $n + m - 1$. Our proposed linear farcaster which includes the first and last weight steps (referred to as **LFD-2**), aims to minimize the combined loss $L_{pred} + \beta \cdot L_{grad}$, where β balances the two losses; β is typically set to a small value, especially when the dimension d is large.

Remark on Efficiency. We demonstrate the efficiency of our approach via the computing of number of FLOPs. Consider performing m updates of $\mathbf{w}_{i+1} \leftarrow \mathbf{w}_i - \alpha_i \nabla L(\mathbf{w}_i)$, where $\mathbf{w}_i \in \mathbb{R}^d$ and $d \gg 1$. In a simple case where the gradient is linear, $\nabla L(\mathbf{w}_i) = \mathbf{C}\mathbf{w}_i - \mathbf{h}$, for matrix \mathbf{C} and vector \mathbf{h} , each update operation requires $2d^2 + 2d$ FLOPs. Consequently, m updates lead to $2md^2 + 2dm$ FLOPs. For our trained model LFS-2 at inference phase, finding m updates demands only $4dm$ FLOPs, while retrieving the m^{th} update alone requires only $4d$ FLOPs which results in significant gains in efficiency. Large-scale deep learning systems frequently involve more complex updates and thus any iterative update scheme will require more FLOPs.

5 Experiments

We implement our approach using PyTorch⁵ and evaluate its performance on both synthetic and real-world weight sequences. The synthetic sequences are derived from our linear regression example, while the real-world sequences are obtained from training two representative deep learning models: a convolutional neural network and DistilBERT. Additional implementation details and training code are provided in Section 1 of the Appendix.

⁵ Our code is publicly available at https://github.com/xshou1990/Less_is_more_weight_farcastering

Table 1: Summary of baseline models.

Model	Functional Form	Output
Introspection	Neural Network $f : \mathbf{X}_T \in \mathbb{R}^{d \times 4} \rightarrow \mathbf{y} \in \mathbb{R}^d$	Point
WNN	Neural Network $f : \psi([\mathbf{X}, d\mathbf{X}]) \in \mathbb{R}^{d \times l} \rightarrow y \in \mathbb{R}^d$	Point
LFN	Neural Network $f : \mathbf{X} \in \mathbb{R}^{d \times n} \rightarrow \mathbf{Y} \in \mathbb{R}^{d \times m}$	Sequence
DLinear	Neural Network $f : \psi(\mathbf{X}) \in \mathbb{R}^{d \times l} \rightarrow \mathbf{Y} \in \mathbb{R}^{d \times m}$	Sequence
LFS	Neural Network $f : \mathbf{X}_T \in \mathbb{R}^{d \times k} \rightarrow \mathbf{Y} \in \mathbb{R}^{d \times m}$	Sequence
LFL	Neural Network $f : \mathbf{X}_n \in \mathbb{R}^{d \times 1} \rightarrow \mathbf{Y} \in \mathbb{R}^{d \times m}$	Sequence
LFD-2	Neural Network $f : \mathbf{X}_{[0,n]} \in \mathbb{R}^{d \times 2} \rightarrow \mathbf{Y} \in \mathbb{R}^{d \times m}$	Sequence

5.1 Synthetic Data Experiments

Synthetic Weight Sequence Generation. We prepare two types of synthetic data to validate our approach:

- **Syn-1:** We generate weight sequences from the least square regression problem in Equation 1. Specifically, we randomly sample optimal weight $\mathbf{w}^* \in \mathbb{R}^3$ from a normal distribution $\mathcal{N}(0, \mathbf{I})$. We similarly sample 100 feature vectors from $\mathcal{N}(0, \mathbf{I})$ to form a feature matrix $\mathbf{D} \in \mathbb{R}^{100 \times 3}$. Correspondingly the generated response vector \mathbf{e} can be computed according to $\mathbf{D}\mathbf{w}^* = \mathbf{e}$. This procedure is similar to previous work on in-context learning [6]. Once we have feature-response pair (\mathbf{D}, \mathbf{e}) , we can formulate a linear regression problem to minimize the least square error. Weight sequences \mathbf{w}_i are generated according to gradient descent and mini-batch stochastic gradient descent respectively.
- **Syn-2:** We similarly generate feature-response pair $(\mathbf{D} \in \mathbb{R}^{100 \times 1}, \mathbf{e} \in \mathbb{R}^{100})$ where this time the regression is performed via a 2-layer fully connected neural network with a total of 31 parameters to minimize, according to Equation 2. We then obtain parameter sequences $\mathbf{w}_i \in \mathbb{R}^{31}$ from training the neural network. Optimizers SGD and Adam are applied to train the neural network respectively.

We generate 200 minimization problems for each synthetic experiment and thus 200 weight sequences are collected. Each one contains 201 time steps including the randomly initialized weight. We carefully choose the learning rate so that it is not so large that it diverges, or so small that it hardly converges. For Syn-1, in the GD update case, we use the reciprocal of the largest eigenvalue of the Hessian matrix multiplied by 0.01; in the SGD update case, we use a batch size of 8 and learning rate of 0.001. For Syn-2, in the SGD case, we set the learning rate of 0.002; and for Adam, 0.005; batch size of 64 is used for both experiments. We split the sequences into 100/50/50 as train/dev/test. For each sequence, we use the first 21 time steps of weights as training sequence \mathbf{X}^i and the remaining 180 time steps for prediction \mathbf{Y}^i . A total of 5 trials for each experiment are recorded.

Baselines. We compare our model LFD-2 with 6 baseline models. A summarized overview of each model is presented in Table 1. Introspection⁶ [28] and WNN⁷ [13] are weight forecasting modules that train on weight sequences and predict a near future weight, thus providing a single time step prediction. DLinear⁸ [35] is a powerful time series long-term forecasting model which was originally proposed to examine the necessity of applying complex transformer models in time series forecasting tasks; a few trainable transformations including MLP and RevIN [16] are applied to better extract temporal and channel-wise features [19].

We also utilize simple baseline models known as LFN and LFL. LFN is a one-layer fully connected neural network designed to minimize Equation 3 by considering all steps in a sequences, while LFL uses solely the very last weight. Additionally, LFS draws inspiration from Introspection by selecting a subset of columns and minimizing the prediction error for each entry based on Equation 4. It is worth noting that, unlike Introspection and WNN, these models produce a sequence of weights for a single inference instance.

Evaluation Metrics. While it is not straightforward to compare the trajectory against some ground truth, we provide checks at various checkpoints for weight step $i \in \{40, 80, 160, 200\}$ for quantitative evaluation similar to that of time series forecasting using mean squared error (MSE) [25, 35]. For qualitative visual evaluation, we provide predicted trajectories for models other than WNN and Introspection, which is not uncommon in evaluating models for time series forecasting tasks.

Table 2: MSE for various models at checkpoints, scaled by 10^4 on Syn-1. Best results are in bold; second best ones are in italics. Standard deviation values are included in parentheses.

Model/Time	GD				SGD			
	40	80	160	200	40	80	160	200
Introspection	.222(.023)	<i>4.63(.41)</i>	45.1(3.8)	405(724)	12.8(1.6)	<i>61.4(13.7)</i>	219(48)	307(45)
WNN	1.75(1.76)	15.9(8.2)	213(62)	278(60)	14.1(.6)	71.8(14.8)	270(29)	412(60)
LFN	.619(.561)	5.11(1.18)	43.8(4.5)	92.4(12.0)	<i>12.7(2.0)</i>	61.7(9.8)	<i>212(25)</i>	339(44)
DLinear	.715(.119)	5.72(.42)	48.1(3.9)	<i>81.8(6.8)</i>	14.2(2.3)	68.6(13.6)	221(43)	<i>302(53)</i>
LFS	<i>.373(.062)</i>	5.34(.65)	60.3(18.7)	205(112)	103(118)	513(536)	1557(1517)	2025(1917)
LFL	193(23)	1337(257)	3787(353)	4594(181)	177(10)	1241(186)	3615(203)	4667(345)
LFD-2	.207(.021)	4.39 (.39)	<i>44.9(5.0)</i>	81.1(9.6)	11.0(1.4)	53.0(8.6)	195(22)	278(18)

Results. The results of the synthetic experiments Syn-1 and Syn-2 are presented in Tables 2 and 3, respectively. For Syn-1, our proposed model LFD-2 outperforms other models by achieving a lower average mean squared error at almost all checkpoints. In the case of Syn-2, LFD-2 demonstrates top-tier performance, comparable to the two most complex models, DLinear and WNN, both in terms

⁶ <https://github.com/muneebshahid/introspection>

⁷ <https://github.com/jjh6297/WNN>

⁸ <https://github.com/plumprc/RTSF/tree/main>

Table 3: MSE for various models at checkpoints, scaled by 10^4 on Syn-2. Best results are in bold; second best ones are in italics. Standard deviation values are included in parentheses.

Model/Time	SGD				Adam			
	40	80	160	200	40	80	160	200
Introspection	13.0(.8)	55.6(40.3)	158(51)	234(50)	12.5(4.9)	53.1(19.9)	85.3(21.4)	104(39)
WNN	1.30(.20)	21.9(2.0)	109(8)	176(66)	4.52(3.53)	9.27(2.26)	<i>44.4(27.9)</i>	74.0(47.8)
LFN	2.20(.50)	<i>25.6(3.50)</i>	123(13)	173(20)	7.88(3.42)	42.6(19.8)	103(25)	129(30)
DLinear	4.99(.21)	31.6(2.3)	126(9)	168(13)	2.60(.25)	<i>12.7(.9)</i>	37.2(3.4)	45.6(4.9)
LFS	5.41(4.59)	43.0(22.0)	166(37)	207(39)	11.1 (5.3)	54.7 (13.6)	108(22)	122(24)
LFL	8.97(.49)	61.7(5.5)	195 (22)	243(27)	7.68(1.86)	43.2(9.7)	101(16)	117(17)
LFD-2	<i>1.83(.40)</i>	26.2(3.6)	<i>123(12)</i>	168(21)	1.13 (.65)	16.3 (11.0)	49.6 (20.4)	<i>60.2(11.1)</i>

of parameter count and the number of input weight columns. Notably, DLinear and WNN are not feasible for training larger neural networks like CNN and DistilBERT, as discussed in the subsequent section. An interesting observation is that the error increases over the predicted steps for all models on our synthetic datasets. This underscores the importance of our farcasting approach and suggests strategies for determining the optimal farcasting length when designing future algorithms.

In addition we provide visual tools to qualitatively evaluate model forecasting performance. We illustrate the full trajectories for 4 farcasting models: DLinear, LFD-2, LFN, and LFS on an exemplar sequence from Syn-1 with GD⁹ and SGD update in Fig. 2. The complexity of these models increases in the order of LFD-2 < LFS < LFN < DLinear. LFD-2 provide comparable and even better prediction to other more complex models with GD update. Trajectories with SGD updates are much more challenging for all models to handle compared to GD, which is typically used in deep learning. LFD-2 and LFN provide better predictions than DLinear and LFS in the SGD setting as they are closer to the ground truth. The trajectories of objective loss from LFD-2 on test subsets show similar converging behavior¹⁰. Moreover, while DLinear is effective for time series forecasting tasks, it may not perform as well for weight prediction problems, where insights from optimization are more critical than periodicity and trend.

5.2 Large Scale Deep Learning Applications

Weight Sequence Generation. We illustrate the efficiency of our model using two examples from large-scale deep learning systems. The first example is a convolutional neural network for image classification, using the standard MNIST dataset¹¹. The CNN model consists of two convolutional layers. We collect 10 sequences of parameter updates using the Adam optimizer while training the CNN model on the MNIST dataset with different random initializations. Each sequence contains 101 weights, and each weight is of 1,199,882 dimensions. We

⁹ See Fig. 1-3 of Appendix for more details.

¹⁰ Figure 5(b) of Appendix

¹¹ <https://github.com/mbjoseph/pytorch-mnist/blob/master/cnn-mnist.ipynb>

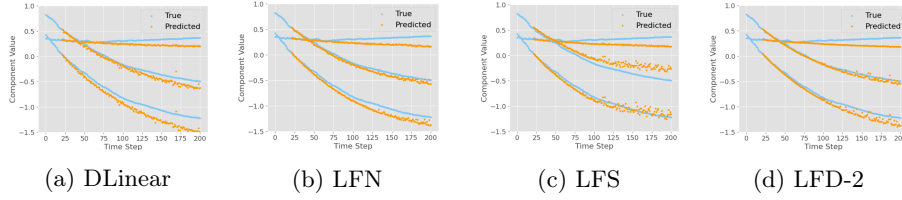


Fig. 2: Examples of forecasted weight sequences and their ground truth are provided for the Syn-1 synthetic data experiment, utilizing the minibatch SGD update. The feature vectors are three-dimensional, and we plot the values for each component across 200 updates for four different models (a)-(d). During the inference phase, the weight sequences from step 0 to 20 are used to forecast the sequences from 21 to 200.

split the sequences into 5-2-3 for train, dev and test subsets. For each sequence, the first 11 steps of weights are used as the training sequence (\mathbf{X}^i), and the remaining 90 time steps are used for prediction (\mathbf{Y}^i). Similarly, we perform text classification using DistilBERT on the Consumer Complaint Dataset from the Consumer Complaint Database¹². Our goal is to classify textual complaints into specific categories, such as Mortgage and Credit Reporting. We utilize an exemplar notebook¹³ that preprocesses the original dataset, uses DistilBERT embeddings, and fine-tunes a sample of the preprocessed dataset. The pretrained DistilBERT model produces embeddings of 66,960,393 dimensions. We generate one sequence of 60 updates using the AdamW optimizer using Amazon SageMaker ml.g5.48xlarge instance¹⁴. We then split the 60 updates into 30-30 for training and testing subsets. Specifically, we train on the first 30 weight subsequence and test on the last 30 weight subsequence. The first 5 steps of weights are used as the training sequence (\mathbf{X}^i), and the remaining 25 time steps are used for prediction (\mathbf{Y}^i).

Results. We present the mean squared error (MSE) results for seven models at various checkpoints in Table 4. The data size demands a high level of memory and computing resources, particularly for large language models like DistilBERT. For instance, storing a sequence of 60 weights requires approximately 40GB of memory. These challenges cause models such as WNN and DLinear with complex transformation operations for training to fail. On the other hand, our proposed model significantly outperforms others in predicting future weights, achieving notably lower MSE. For example, forecasting the 30th step using DistilBERT results in an order of magnitude improvement over most models while remaining the same order of magnitude of computing time.

¹² <https://catalog.data.gov/dataset/consumer-complaint-database>

¹³ <https://github.com/vilcek/fine-tuning-BERT-for-text-classification/blob/master/02-data-classification.ipynb>

¹⁴ <https://aws.amazon.com/sagemaker/pricing/>

Table 4: MSE on two deep learning training applications: CNN for MNIST classification and DistilBERT for complaint classification, scaled by 10^4 . "*" indicates model outputs a single time step. Experiments were conducted on the same machine on a private server.

Model	CNN with Adam				DistilBERT with AdamW				sec/epoch
	20	40	80	100	15	20	25	30	
Introspection	9.59	.646	7.58	1.68	.229	.723	.264	.528	1.28*
WNN	5.11	10.9	1.92	7.40	NA	NA	NA	NA	>10*
LFN	.183	.629	1.37	1.59	.109	.0485	.147	.300	6.75
DLinear	NA	NA	NA	NA	NA	NA	NA	NA	>10
LFS	.292	.749	1.37	1.78	.664	.129	.0343	.146	6.89
LFL	.179	.693	1.39	1.68	.0764	.134	.377	.0288	3.19
LFD-2	.137	.599	1.29	1.60	.0955	.194	.0803	.00985	9.42

5.3 Ablational Experiments

We address the following two questions via ablatinal experiments in this section.

Table 5: Farcast with loss, scaled by 10^4 .

Experiment	40	80	160	200
Syn-1 GD	.196(.015)	4.42 (.47)	43.8 (5.5)	78.0(6.7)
Syn-1 SGD	10.9(1.3)	52.9(8.5)	194(23)	278(18)
Syn-2 SGD	1.74(.12)	24.7(1.5)	126(9)	175(5)
Syn-2 Adam	.799(.229)	13.5(5.7)	37.4(4.4)	54.7(11.1)

Can training loss help the forecast? In ML and DL, the loss at each update is readily accessible, typically through a forward pass in deep learning. We are interested in exploring whether augmenting the weights with the loss values, i.e., using (\mathbf{w}_i, l_i) , can improve the prediction of future weights. Therefore, we incorporated such losses in our Syn-1 and Syn-2 experiments. The results in Table 5 indicate that our model, LFD-2, does not benefit significantly from adding these losses. A possible explanation is that a linear mapping may not be sufficient for effectively learning the relationship between the loss function and the weights. Moreover, while the loss may not directly enhance the prediction of weights, it could directly benefit the prediction of future losses. More complex architectures or weight topologies might be necessary to leverage loss information for guiding the learning of weights.

Table 6: Farcast with $\{\mathbf{w}_0, \mathbf{w}_1\}$, scaled by 10^4 .

Experiment	40	80	160	200
Syn-1 GD	247(15)	1625(85)	4765(220)	6017(190)
Syn-1 SGD	230(14)	1486(103)	4419(188)	5710(279)
Syn-2 SGD	12.6(1.1)	75.2(4.4)	235(24)	294(32)
Syn-2 Adam	14.5(2.6)	72.5(15.2)	163(27)	186(29)

How many gradient steps are necessary for good predictive performance? When dealing with models that have a large number of parameters, such as large language models (LLMs) like DistilBERT, it is important to limit the number of gradient steps (or backward passes) needed to reduce training cost. To investigate this, we conducted experiments using the Syn-1 and Syn-2 datasets as proxies for LLMs such as DistilBERT. Specifically, we used $\{\mathbf{w}_0, \mathbf{w}_1\}$ to forecast \mathbf{w}_i for $i \in \{2, 3, \dots, 200\}$. The results from our model LFD-2, shown in Table 6, indicate that linear regression with a least squares fit performs significantly worse than that trained by a neural network. One possible explanation, as suggested by previous work [28], is that the weights in neural networks generally do not change significantly over time. While using only the first two steps and forecasting a limited number of steps ahead is not overly detrimental, performance does decline compared to training with the first 21 steps in Table 2 and 3. In practice, an analyst may choose to train for a certain number of steps and then use our model LFD-2 for forecasting, depending on computational resources.

6 Conclusion

We address the computational challenges of training ML and DL systems with a novel framework. Our proposed approach adapts long-term time series forecasting techniques to neural network weight predicting problem and we further improve efficiency by using minimal number of previous steps while maintaining superior accuracy. Adaptive weight prediction or simplification are potential directions for future work. While we validate our method on CNNs and DistilBERT, scaling to larger architectures like full BERT, or even GPT models remains an exciting direction.

References

1. Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M.W., Pfau, D., Schaul, T., Shillingford, B., De Freitas, N.: Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems* **29** (2016)
2. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Nee-lakantan, A., Shyam, P., Sastry, G., Askell, A., et al.: Language models are few-shot learners. *Advances in neural information processing systems* **33**, 1877–1901 (2020)

3. Chen, R.T., Rubanova, Y., Bettencourt, J., Duvenaud, D.K.: Neural ordinary differential equations. *Advances in neural information processing systems* **31** (2018)
4. Chevillon, G.: Direct multi-step estimation and forecasting. *Journal of Economic Surveys* **21**(4), 746–785 (2007)
5. Ding, Y., Huang, Z., Shou, X., Guo, Y., Sun, Y., Gao, J.: Architecture-aware learning curve extrapolation via graph ordinary differential equation. *arXiv e-prints* pp. arXiv-2412 (2024)
6. Garg, S., Tsipras, D., Liang, P.S., Valiant, G.: What can transformers learn in-context? a case study of simple function classes. *Advances in Neural Information Processing Systems* **35**, 30583–30598 (2022)
7. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. pp. 249–256. *JMLR Workshop and Conference Proceedings* (2010)
8. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: *Proceedings of the IEEE international conference on computer vision*. pp. 1026–1034 (2015)
9. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. pp. 770–778 (2016)
10. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015)
11. Huang, Z., Sun, Y., Wang, W.: Learning continuous system dynamics from irregularly-sampled partial observations. *Advances in Neural Information Processing Systems* **33**, 16177–16187 (2020)
12. Huang, Z., Sun, Y., Wang, W.: Coupled graph ode for learning interacting system dynamics. In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. pp. 705–715 (2021)
13. Jang, J., Yun, W.h., Kim, W.H., Yoon, Y., Kim, J., Lee, J., Han, B.: Learning to boost training by periodic nowcasting near future weights. In: *International Conference on Machine Learning*. pp. 14730–14757. *PMLR* (2023)
14. Jia, X., Song, S., He, W., Wang, Y., Rong, H., Zhou, F., Xie, L., Guo, Z., Yang, Y., Yu, L., et al.: Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205* (2018)
15. Khailany, B.: Accelerating chip design with machine learning. In: *Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD*. pp. 33–33 (2020)
16. Kim, T., Kim, J., Tae, Y., Park, C., Choi, J.H., Choo, J.: Reversible instance normalization for accurate time-series forecasting against distribution shift. In: *International Conference on Learning Representations* (2021)
17. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014)
18. Knyazev, B., Drozdal, M., Taylor, G.W., Romero Soriano, A.: Parameter prediction for unseen deep architectures. *Advances in Neural Information Processing Systems* **34**, 29433–29448 (2021)
19. Li, Z., Qi, S., Li, Y., Xu, Z.: Revisiting long-term time series forecasting: An investigation on linear mapping. *arXiv preprint arXiv:2305.10721* (2023)
20. Loshchilov, I., Hutter, F.: Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983* (2016)
21. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101* (2017)

22. Luo, D., Wang, X.: Moderntcn: A modern pure convolution structure for general time series analysis. In: The Twelfth International Conference on Learning Representations (2024)
23. Luo, X., Yuan, J., Huang, Z., Jiang, H., Qin, Y., Ju, W., Zhang, M., Sun, Y.: Hope: High-order graph ode for modeling interacting dynamics (2023)
24. Menghani, G.: Efficient deep learning: A survey on making deep learning models smaller, faster, and better. *ACM Computing Surveys* **55**(12), 1–37 (2023)
25. Nie, Y., Nguyen, N.H., Sinthong, P., Kalagnanam, J.: A time series is worth 64 words: Long-term forecasting with transformers. *arXiv preprint arXiv:2211.14730* (2022)
26. Rodriguez, P., Gonzalez, J., Cucurull, G., Gonfaus, J.M., Roca, X.: Regularizing cnns with locally constrained decorrelations. *arXiv preprint arXiv:1611.01967* (2016)
27. Sanh, V., Debut, L., Chaumond, J., Wolf, T.: Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* (2019)
28. Sinha, A., Sarkar, M., Mukherjee, A., Krishnamurthy, B.: Introspection: Accelerating neural network training by learning weight evolution. *arXiv preprint arXiv:1704.04959* (2017)
29. Taieb, S.B., Hyndman, R.J., et al.: Recursive and direct multi-step forecasting: the best of both worlds, vol. 19. Department of Econometrics and Business Statistics, Monash Univ. (2012)
30. Tan, M., Le, Q.: Efficientnet: Rethinking model scaling for convolutional neural networks. In: International conference on machine learning. pp. 6105–6114. PMLR (2019)
31. Wang, Y., Singh, A.: Provably correct algorithms for matrix column subset selection with selectively sampled data. *Journal of Machine Learning Research* **18**(156), 1–42 (2018)
32. Xue, W., Zhou, T., Wen, Q., Gao, J., Ding, B., Jin, R.: Make transformer great again for time series forecasting: Channel aligned robust dual transformer. *arXiv preprint arXiv:2305.12095* (2023)
33. Ye, J., Zhang, W., Yi, K., Yu, Y., Li, Z., Li, J., Tsung, F.: A survey of time series foundation models: Generalizing time series representation with large language mode. *arXiv preprint arXiv:2405.02358* (2024)
34. Zang, C., Wang, F.: Neural dynamics on complex networks. In: Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining. pp. 892–902 (2020)
35. Zeng, A., Chen, M., Zhang, L., Xu, Q.: Are transformers effective for time series forecasting? In: Proceedings of the AAAI conference on artificial intelligence. vol. 37, pp. 11121–11128 (2023)
36. Zhang, C., Ren, M., Urtasun, R.: Graph hypernetworks for neural architecture search. *arXiv preprint arXiv:1810.05749* (2018)