Article

Not peer-reviewed version

A Globally Optimal Alternative to MLP

Zheng Li , Jerry Cheng , Huanying Gu *

Posted Date: 16 June 2025

doi: 10.20944/preprints202506.1335.v1

Keywords: global minimum; discrete optimization; Lagrange mesh

 doi:10.20944/preprints202506.1335.v1

Article

A Globally Optimal Alternative to MLP

Zheng Li [1,†,‡], Jerry Cheng [2,‡] and Huanying Gu [2,*]

[1] Department of Computer Science, New York Institute of Technology, New York, NY 10023
[2] Department of Computer Science, New York Institute of Technology, New York, NY 10023
* Correspondence: hgu03@nyit.edu
† Current address: New York Institute of Technology.
‡ These authors contributed equally to this work.

Abstract: In deep learning, achieving the global minimum poses a significant challenge, even for relatively simple architectures such as Multi-Layer Perceptrons (MLPs). To address this challenge, we visualized model states at both local and global optima, thereby identifying the factors that impede the transition of models from local to global minima when employing conventional model training methodologies. Based on these insights, we propose the Lagrange Regressor (LReg), a framework that is mathematically equivalent to MLPs. Rather than updates via optimization techniques, LReg employs a mesh-refinement-coarsening (discrete) process to ensure the convergence of the model's loss function to the global minimum. LReg achieves faster convergence and overcomes the inherent limitations of neural networks in fitting multi-frequency functions. Experiments conducted on large-scale benchmarks including ImageNet-1K (image classification), GLUE (natural language understanding), and WikiText (language modeling) show that LReg consistently enhances the performance of pre-trained models, significantly lowers test loss, and scales effectively to big data scenarios. These results underscore LReg's potential as a scalable, optimization-free alternative for deep learning in large and complex datasets, aligning closely with the goals of innovative big data analytics.

Keywords: global minimum; discrete optimization; Lagrange mesh

1. Introduction

Current research suggests that in sufficiently large neural networks, most local minima in associated loss functions have relatively small values. Consequently, the requirement of locating a global minimum becomes less crucial, as it is more significant to identify a point within the parameter space with a low loss, though not necessarily minimal [1–4]. This perspective has been substantiated through various studies on large language models. These studies demonstrate that independent training of an identical model often results in final cost function values that display minimal variation. This observation suggests that the differences between most local minima, including the global minimum, are potentially negligible. Additionally, scaling laws [5,6] establish a relationship between the number of model parameters and the (minimum) value of cost function.

Nonetheless, these regular patterns have yet to be thoroughly investigated, while they have been predominantly inferred based on experimental results. For example, in the case of scaling laws, the constants in the empirical formula can be approximated solely through experimental data. Furthermore, practitioners face significant challenges in determining whether a model's cost function has attained the global minimum during training, as there is a lack of research providing methodologies to calculate the lower bound of a given cost function or to visualize a model's state upon reaching the global minimum.

In this study, we begin by establishing the equivalence between MLPs and Lagrange linear interpolation functions. Subsequently, we explore the performance of MLPs in regression tasks, with a visualization of the loss function's states at both local minima (Figure 1a) and global minima (Figure 1d). *Our analysis demonstrates that conventional training methods, which depend on continuous parameter*

 doi:10.20944/preprints202506.1335.v1

updates, are fundamentally inadequate for guiding a loss function from a local minimum to a global minimum. To address this issue, we propose a Mesh-Refinement-Coarsening (MRC) loop (Figure 1 b & c) for updating model parameters, inspired by Finite Element Method (FEM) theory [7]. This algorithm reformulates the MLP as an equivalent Lagrange linear interpolation function, and we refer to the alternative model of MLP as Lagrange Regressor (LReg).

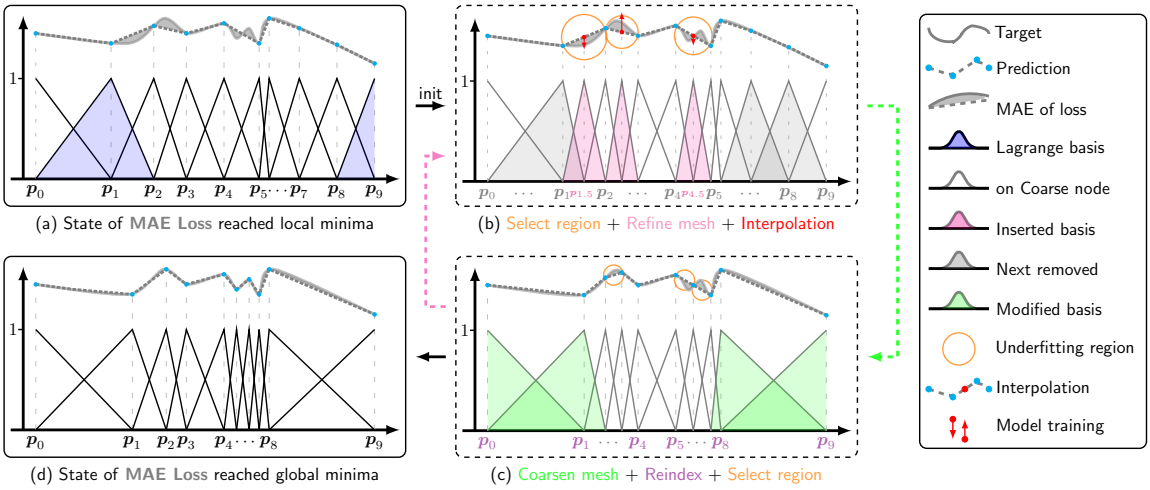

Figure 1. Transition a loss function from a local minimum to a global minimum. (a) Local Minimum: The MLP, based on linear layers and ReLU activations, is represented as a segmented line. Slight adjustments to this segmented line increase the area of shaded region, which represents the Mean Absolute Error (MAE); (b) Refinement: Detect underfitting regions of the MLP and insert midpoints to reduce the MAE; (c) Coarsening: Remove nodes with minimal impact on the MAE, decreasing the number of model parameters back to its original count. Scale up the corresponding Lagrange basis and return to Step b; (d) Global Minimum: The shaded area representing the MAE is minimized.

LReg serves as an alternative to MLPs, with the ability to ensure a rapid convergence of a loss function towards its global minimum. It is applicable to a range of recognition domains, including regression, image classification, image super-resolution, text classification, and language modeling. Furthermore, our experimental results on large datasets, such as ImageNet, demonstrate that LReg, when explained as a Parameter-Efficient Fine-Tuning (PEFT) method, it promotes further convergence of the validation loss function in large pre-trained models towards global minima (see Figure 5) and enhances the model performance. In contrast to other PEFT methods, LReg has a versatile structure that is not limited to transformer-based architectures, thereby increasing its applicability. Our contributions are as follows:

1. Investigating and visualizing the challenge encountered by the loss function of an MLP in overcoming to escape local minima to attain the global minimum.
2. Deriving the explicit scaling law for MLPs through error bounds and conducting experimental validation on regression tasks.
3. Overcoming the limitations of neural networks in fitting multi-frequency data, thereby making LReg more versatile for complex applications.
4. Applying LReg as a PEFT method for large models (not confined to transformer architectures) to investigate the potential of guiding their loss functions toward global minima, consequently further reducing test loss and improving the performance of pre-trained models.

 doi:10.20944/preprints202506.1335.v1

2. Methods

2.1. Preliminary: Function Fitting via MLP

Mathematically, the composition of any piecewise linear functions remains piecewise linear. Furthermore, any piecewise linear function $f(x)$ can be represented as a linear combination of Lagrange basis functions:

$$f(x) = \sum_i c_i \psi_i(x), \tag{1}$$

where c_i is a coefficient of the basis function $\psi_i(x)$.

It is noteworthy that both linear layers and ReLU activations are piecewise linear functions. Consequently, an MLP composed of these elements inherently constitutes a piecewise linear mapping. For such an MLP, it is always possible to determine a set of coefficients $\{c_1, c_2, \dots\}$ and basis functions $\{\psi_1, \psi_2, \dots\}$ such that the aforementioned equation can represent the MLP.

The geometry of piecewise linear functions is straightforward: in one-dimensional space, these functions appear as segmented lines, while in two-dimensional space, they form triangulated surfaces. Figure 1a illustrates the performance of an MLP on a function-fitting task. In this example, the MLP fits the function by the dashed segmented line, while the ground truth is represented by a solid curve. The area of the shaded region between them represents the MAE. Notably, any minor alteration to any node on the segmented line will result in an increase in the MAE, because the MAE is at a **local minimum** in this configuration.

2.2. Optimization: Discrete vs. Continuous Processes

Upon examining Figure 1 closely, the transitioning from the piecewise linear function depicted in Figure 1a to that in Figure 1d requires two key modifications: **(1)** relocation of nodes $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, and **(2)** the shift of nodes $\boldsymbol{p}_7$ and $\boldsymbol{p}_8$ into the interval between $\boldsymbol{p}_4$ and $\boldsymbol{p}_5$. This is a **discrete updating** process as it requires reordering $(\boldsymbol{p}_4, \boldsymbol{p}_5, \boldsymbol{p}_6, \boldsymbol{p}_7, \boldsymbol{p}_8, \boldsymbol{p}_9)$ to $(\boldsymbol{p}_4, \boldsymbol{p}_7, \boldsymbol{p}_8, \boldsymbol{p}_5, \boldsymbol{p}_6, \boldsymbol{p}_9)$, followed by adjusting the positions of $\boldsymbol{p}_7$ and $\boldsymbol{p}_8$. The former adjustment might be achievable through gradient-based methods, whereas the latter is almost impossible. This difficulty arises while updating parameters continuously with first-order adaptive learning rate methods. With the learning rate inevitably diminishing to very small values, the parameter updates become negligible, reducing the likelihood of the model undergoing the significant changes required for the transition from Figure 1a to Figure 1d.

2.3. Lagrange Basis Function

Our **MRC** algorithm is designed for **discrete updates** of model parameters. However, it is not directly applicable to the training of MLPs. In order to employ it effectively, we first need to reformulate the MLP as a linear combination of basis functions, as shown in Eqn. (1). *The Lagrange basis functions are defined as tent-shaped linear functions that take a value of 1 at their specified nodes and 0 at all other nodes.* In Appendix A.2, we derive the formula for the Lagrange basis function in detail. For the one-dimensional case, when fitting a continuous function $F : \mathbb{R} \to \mathbb{R}$ using an MLP, the Lagrange basis function can be expressed as:

$$\psi_i(\boldsymbol{x}) = \min\left(\frac{\mathrm{ReLU}(\boldsymbol{x} - \boldsymbol{p}_{i-1})}{\boldsymbol{p}_i - \boldsymbol{p}_{i-1}}, \frac{\mathrm{ReLU}(\boldsymbol{p}_{i+1} - \boldsymbol{x})}{\boldsymbol{p}_{i+1} - \boldsymbol{p}_i}\right).$$

For example, ψ_1 and ψ_9 are shaded blue in Figure 1a for illustrative purposes. In FEM, a **mesh** refers to a network of interconnected points, called **nodes** (e.g., ten points $\{\boldsymbol{p}_0, \boldsymbol{p}_1, \dots\}$ in Figure 1a), which divide a complex geometry into smaller, simpler **simplices** (e.g., nine intervals $\{(\boldsymbol{p}_0, \boldsymbol{p}_1), (\boldsymbol{p}_1, \boldsymbol{p}_2), \dots\}$ in Figure 1a).

When fitting a continuous function $F : \mathbb{R}^d \to \mathbb{R}$ using an MLP, the formulation for the Lagrange basis becomes substantially more intricate. This necessitates an understanding of the terminologies associated with **Delaunay Triangulation**. In general scenarios, the mathematical expressions of the Lagrange basis are described in Appendix A.2, and the visualizations are provided in Appendix A.5.

 doi:10.20944/preprints202506.1335.v1

2.4. Mesh-Refinement-Coarsening Loop

Inspired by the FEM, we designed the MRC loop to update model parameters. The process begins with *identifying underfitting regions*, where the fitted value significantly deviates from the target. As shown in Figure 1b, the model exhibits MAE in the elements (p_1, p_2), (p_2, p_3), and (p_4, p_5); Next, we perform *mesh refinement* by inserting fine nodes $p_{1.5}$, $p_{2.5}$, and $p_{4.5}$ at the center of these regions (For the two-dimensional case $F : \mathbb{R}^2 \to \mathbb{R}$, we provide a visual example of fine node insertion in Appendix A.5) and denote others as coarse nodes. The corresponding coefficients $c_{1.5}$, $c_{2.5}$, and $c_{4.5}$ are computed via *interpolation* to construct the scaled-up model; Subsequently, we further update these new coefficients to continuously reduce MAE using model training or the least squares method (see Appendix A.5 for details); The scaled-up model will have a reduced MAE value but with a bigger model size. For example, there are three more basis functions involved. In order to keep model size unchanged, we perform *coarse node removal* to scale the model down after maintaining the MAE value as much as possible. In Figure 1c, three nodes are chosen to be removed, the adjacent simplices are merged to form a *scaled Lagrange basis function* (see Figure 1c); We then reindex the nodes, denoting them as coarse nodes to start the next iteration of the loop. The following transformations illustrate the MRC loop in Figure 1, detailing the key steps in mesh refinement, coefficient interpolation, mesh coarsening, and reindexing.

[Mesh Refinement] Insert three nodes:

$$(p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9) \to (p_0, p_1, p_{1.5}, p_2, p_{2.5}, p_3, p_4, p_{4.5}, p_5, p_6, p_7, p_8, p_9).$$

[Mesh Refinement] Add three basis functions:

$$(\psi_0, \psi_1, \psi_2, \psi_3, \psi_4, \psi_5, \psi_6, \psi_7, \psi_8, \psi_9) \to (\psi_0, \psi_1, \psi_{1.5}, \psi_2, \psi_{2.5}, \psi_3, \psi_4, \psi_{4.5}, \psi_5, \psi_6, \psi_7, \psi_8, \psi_9).$$

[Interpolation] Compute three coefficients:

$$(c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9) \to (c_0, c_1, c_{1.5}, c_2, c_{2.5}, c_3, c_4, c_{4.5}, c_5, c_6, c_7, c_8, c_9).$$

[Mesh Coarsening] Remove three coarse nodes:

$$(p_0, p_1, p_{1.5}, p_2, p_{2.5}, p_3, p_4, p_{4.5}, p_5, p_6, p_7, p_8, p_9) \to (p_0, p_{1.5}, p_2, p_{2.5}, p_3, p_4, p_{4.5}, p_5, p_6, p_9).$$

[Mesh Coarsening] Merge simplices that are adjacent to the removed nodes:

$$(p_0, p_1), (p_1, p_{1.5}) \to (p_0, p_{1.5})(p_6, p_7), (p_7, p_8), (p_7, p_9) \to (p_8, p_9).$$

[Mesh Coarsening] Keep the number of basis functions:

$$(\psi_0, \psi_1, \psi_{1.5}, \psi_2, \psi_{2.5}, \psi_3, \psi_4, \psi_{4.5}, \psi_5, \psi_6, \psi_7, \psi_8, \psi_9) \to (\psi_0, \psi_{1.5}, \psi_2, \psi_{2.5}, \psi_3, \psi_4, \psi_{4.5}, \psi_5, \psi_6, \psi_9).$$

[Node Reindexing]:

$$(p_0, p_{1.5}, p_2, p_{2.5}, p_3, p_4, p_{4.5}, p_5, p_6, p_9) \to (p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9).$$

These transformations ensure the MRC loop efficiently adapts the model structure, reducing underfitted regions while maintaining computational efficiency.

Figure 1 shows that as this loop progresses, the underfitting regions gradually shrink, and the MAE consistently decreases. This process significantly deviates from first-order adaptive learning rate methods. Our methodology utilizes discrete operations to update model parameters, check Algorithm 1 for details.

 doi:10.20944/preprints202506.1335.v1

Algorithm 1 Mesh-Refinement-Coarsening Loop

while global minimum not reached **do**
  Identify top-k underfitted regions.
  Refine the mesh by adding midpoints.
  Compute coefficients of new basis via interpolation.
  Adjust coefficients using least squares or training.
  Remove redundant coarse nodes.
  Reindex all nodes.
end while

2.5. Scaling Law and Error-Bound Formula

Our approach demonstrates significant interpretability, supported by the universal approximation theorem and our quantitative experimental results. In the FEM theory, the number of parameters N is proportional to the number of simplices n_t in the triangular mesh (refer to Appendix A.2). Since $(n_t/d!)^{1/d} = O(h^{-1})$, the error bound formula for fitting function $F : \mathbb{R}^d \to \mathbb{R}$ is derived as:

$$L(N) = |f(\boldsymbol{x}) - F(\boldsymbol{x})|_{x\in\Omega} = O(\max_{\boldsymbol{\xi}\in\Omega} \|\nabla F(\boldsymbol{\xi})\| \cdot n_t^{-1/d}) = O(N^{-1/d}),$$

where $h = \max_i \max_j \mathbb{1}_{\text{dist}(\boldsymbol{p}^{(i)},\boldsymbol{p}^{(j)})=1} \cdot \|\boldsymbol{p}^{(i)} - \boldsymbol{p}^{(j)}\|$ represents the maximum length of mesh edges (see Appendix A), and Ω is the domain over which the approximation occurs.

Our experiment shown in Figure 5 demonstrates that we perfectly predicted the explicit scaling law.

2.6. LReg Applied to Large Models

Based on the interpolation formula Eqn. (1), we propose a Lagrange Regressor (LReg) which is equivalent to MLP. LReg adheres to the universal approximation theorem and exhibits strong interpretability for low-dimensional recognition tasks (see Sec. 3.1). However, as the data dimension d increases, the output dimension of LReg grows factorially ($O(d!)$) as described in Sec. 2.5. This exponential growth makes it computationally expensive for high-dimensional data, such as large images, thereby imposing significant challenges in environments with constrained computational resources.

To address this issue, LReg can be incorporated as a module within PEFT methodologies. As shown in Figure 2, consider the target $F(\boldsymbol{x})$ and a pre-trained model $f(\boldsymbol{x})$. When $\tilde{f}(\boldsymbol{x})$ is well trained, the residual $F(\boldsymbol{x}) - \tilde{f}(\boldsymbol{x})$ remains nearly flat around the value of 0 throughout the input space. This sparsity in the residual's support set suggests that dimensionality reduction on features could serve as an effective strategy for applying LReg to high-dimensional inputs.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 June 2025 doi:10.20944/preprints202506.1335.v1

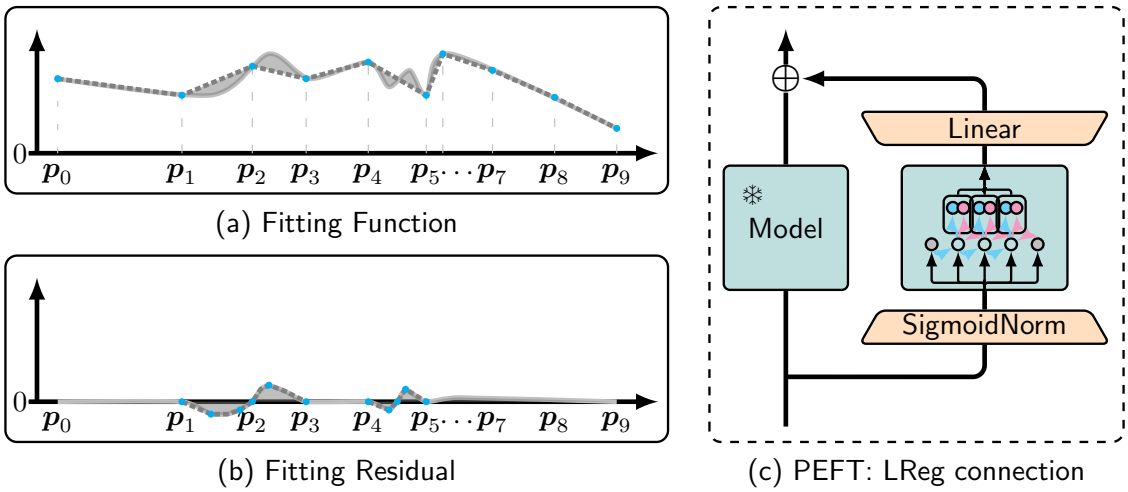

Figure 2. Fitting Function vs. Fitting Residual. (a) Fitting a function $F(x)$ by training an model from scratch; (b) Fitting the residual $F(x) - \tilde{f}(x)$ using the pre-trained model $\tilde{f}(x)$ from (a); (c) LReg is applied as a shortcut connection to fit the residual $F(x) - \tilde{f}(x)$, where the pre-trained model $\tilde{f}(x)$ is kept frozen.

In Figure 1c, we use a SigmoidNorm layer to reduce the input dimensionality:

$$\boldsymbol{x} \leftarrow \text{Sigmoid}(\text{PCA}(\boldsymbol{x})).$$

The built-in PCA module can be obtained through standard unsupervised methods or by training. LReg is then applied to learn the residual:

$$F(\boldsymbol{x}) - \tilde{f}(\boldsymbol{x}) \approx \sum_i c_i \psi_i(\boldsymbol{x}). \tag{2}$$

Subsequently, the composed $\boldsymbol{x}$ is converted to $\{\psi_i(\boldsymbol{x})\}_i$ by the encoder of LReg. Finally, the linear combination $\sum_i c_i \psi_i(\boldsymbol{x})$ is completed via the Linear layer.

In this scenario, LReg enables the loss function of large pre-trained models to move from local minima. Experiments demonstrate that LReg connection further reduces cross-entropy loss and significantly improves the performance of pre-trained models within a single training epoch (see Table 2). This approach effectively fine-tunes the model while preserving computational efficiency.

3. Results

We begin by training LReg from scratch to validate the explicit scaling law in Sec. 2.5 and identify the limitations of conventional models in fitting multi-frequency data (Sec. 3.2). The exploration of model fine-tuning for achieving reduced loss function values and enhanced performance is discussed in Sec. 3.3. For extending the PEFT method to non-transformer-based models via LReg connection, along with comparison results, refer to Sec. 3.4. Implementation details are provided in Appendix A, while additional applications are discussed in Appendix C.

3.1. Scaling Law and Addressing Regression Challenges

Figure 3 illustrates the performance of the LReg method with varying numbers of model parameters. In these experiments, LReg is trained to approach convergence on fitting the mapping function $F : \mathbb{R}^d \to \mathbb{R}$. As shown, the $L(N)$, denoted as test MAE loss, is observed to be proportional to the power of the number of model parameters, $N^{-1/d}$, and the inverse power of the number of simplices, $n_t^{-1/d}$, on the triangulation mesh. This observation is in perfect agreement with the derived error-bound formula in Sec. 2.5.

 doi:10.20944/preprints202506.1335.v1

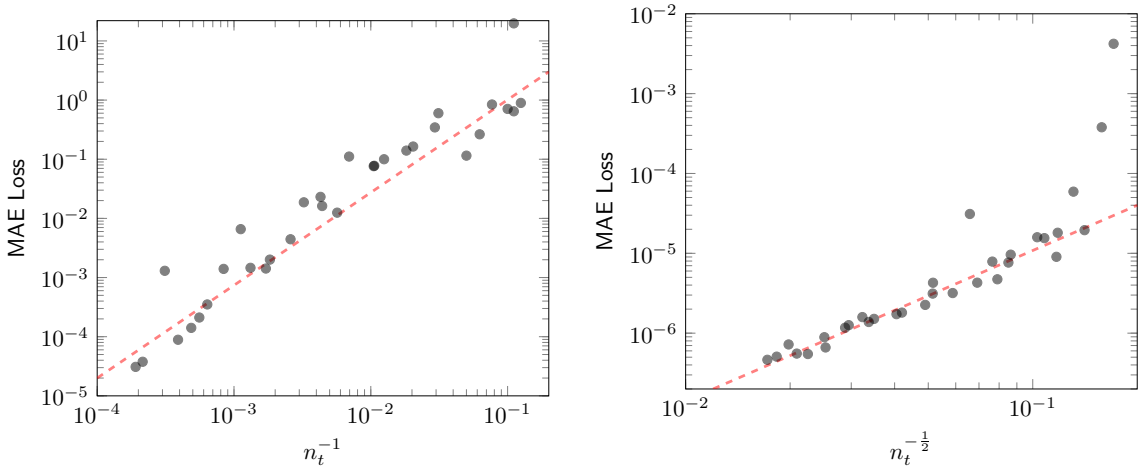

Figure 3. Left - Results of 32 experiments fitting the 1-dimensional function $y = \sin\frac{1}{x}$, with each gray point representing an experiment. The figure shows the relationship between n_t^{-1} and test MAE loss. Right - Results of 32 experiments fitting the 2-dimensional function $y = \sum_{i=1}^{2} \sin\frac{x_i}{2\pi}$, with each gray point representing an experiment. The figure shows the relationship between $n_t^{-1/2}$ and test loss.

Notably, the data learned by the model in Figure 3, represented as $\{(x,y)|\frac{1}{X} \sim U(0.02,1.0), Y \sim \mathcal{N}(\sin\frac{1}{x},0.01)\}$, exhibits multi-frequency characteristics that present significant challenges for conventional neural networks to learn effectively. **It is worth noting that** although distribution $\{(x,y)|\frac{1}{X} \sim U(0.02,1.0), Y \sim \mathcal{N}(\sin\frac{1}{x},0.01)\}$ and distribution $\{(x,y)|X \sim U(1,50), Y \sim \mathcal{N}(\sin x,0.01)\}$ are equivalent in mathematics, **the numerical results of function fitting are completely different**. In our open-source GitHub repository, we provide such a demo: for the neural network without LReg connection, the model fits the latter well, but cannot fit the former anyway.

Traditional regressors often struggle with overfitting, while neural networks address this but perform poorly when fitting multi-frequency functions, a phenomenon known as the *frequency principle* [8]. As shown in Figure 4, Support Vector Regression [9] fits dataset $\mathbb{B} = \{(x,y)|\frac{1}{X} \sim u(0.02,0.5), y = \sin\frac{1}{x}\}$ but overfits on dataset $\mathbb{A}$, whereas neural networks perform well on $\mathbb{A} = \{(x,y)|X \sim u(0,\frac{\pi}{4}), Y \sim N(\sin 8x, |\cos 8x|)\}$ but underfit on $\mathbb{B}$. *Neither MLPs, CNNs, nor Transformers can effectively fit sharp transitions in dataset* $\mathbb{B}$*, often treating these regions as noise to avoid overfitting.* Importantly, no existing model can fit the challenging dataset $\mathbb{C} = \{(x,y)|\frac{1}{X} \sim u(0.02,0.5), Y \sim N(\sin\frac{1}{x}, 0.5x^2)\}$.

LReg addresses this gap by combining the strengths of traditional regressors and neural networks. It leverages a multiscale mesh to fit high-frequency regions using fine simplices adaptively. Figure 4 (right) demonstrates the LReg's unique ability to handle dataset $\mathbb{C}$, a task no other model can achieve. In the next section, we further illustrate its gradual fitting process during training. Comparison experiments in Table 1 show that the LReg consistently achieves high R^2 scores across all test sets, demonstrating its robustness and effectiveness, even in high-noise and multi-frequency scenarios.

 doi:10.20944/preprints202506.1335.v1

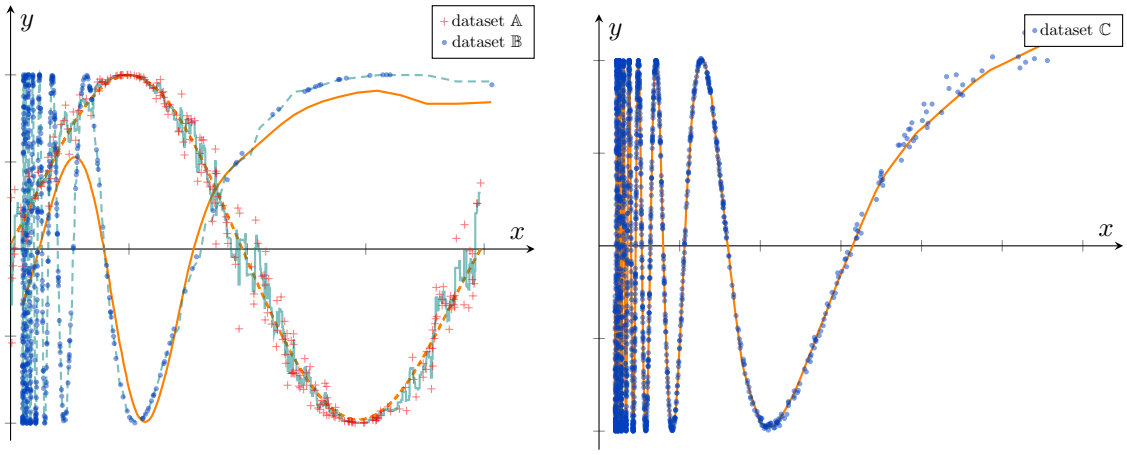

Figure 4. Left - Performance of traditional regressors and neural networks on high-noise dataset $\mathbb{A}$ and multi-frequency dataset $\mathbb{B}$. The dashed teal curve shows that traditional regressors (e.g., Support Vector Regression) succeed on $\mathbb{B}$ but overfit $\mathbb{A}$ (solid teal curve). In contrast, neural networks (e.g., MLP) fit $\mathbb{A}$ well (dashed orange curve) but underfit $\mathbb{B}$ (solid orange curve). Right - The LReg demonstrates exceptional adaptability, successfully learning dataset $\mathbb{C}$, which incorporates both high noise and multi-frequency features.

3.2. Comparison with Regressors: Training from Scratch

As mentioned earlier, neural networks often struggle to fit multi-frequency datasets effectively. Therefore, our primary focus is comparing the LReg with traditional regressors. To evaluate the effectiveness and generalization of the LReg, we have devised four diverse datasets, each generated from distinct probability distributions:

1. $\mathbb{A}^1$: Generated from the distribution $\{(x,y)|X \sim U(-\pi,\pi), Y \sim \mathcal{N}(\sin x, \frac{1}{5}\cos^2 x)\}$, with 1000 training examples and 200 test examples. The LReg was trained with a learning rate of 0.1.
2. $\mathbb{B}^1$: Generated from the distribution $\{(x,y)|\frac{1}{X} \sim U(0.02,1.0), Y \sim \mathcal{N}(\sin\frac{1}{x}, 0.01)\}$, with 1000 training examples and 200 test examples. We trained the LReg with a learning rate of 0.9.
3. $\mathbb{A}^2$: Generated from the distribution $\{(\boldsymbol{x},y)|\boldsymbol{X}_i \sim U(-\pi,\pi), Y_i \sim \mathcal{N}(\sin \boldsymbol{x}_i, \frac{1}{10}\cos^2 \boldsymbol{x}_i),$ $Y = \frac{1}{2}(Y_1 + Y_2)\}$, with 7,500 training examples and 1,500 test examples. The LReg was trained with a learning rate of 0.1.
4. $\mathbb{B}^2$: Generated from the distribution $\{(\boldsymbol{x},y)|\frac{1}{\boldsymbol{X}_i} \sim U(0.05,0.5), Y_i \sim \mathcal{N}(\sin\frac{1}{x_i}, 0.01),$ $Y = \frac{1}{2}(Y_1 + Y_2)\}$, with 50,000 training examples and 10,000 test examples. Training utilized a learning rate of 0.9.

Table 1 displays the coefficient of determination (R^2) scores for the LReg and traditional regressors [9–18] across fitting the four datasets. The LReg consistently achieves high R^2 scores across all test sets, demonstrating the effectiveness of the InterpolationNet on both high-noise and multi-frequency datasets. Furthermore, the minimal gap between training and test set evaluations underscores the robustness of the LReg, indicating its capability of generalization.

 doi:10.20944/preprints202506.1335.v1

Table 1. A comprehensive comparison between the LReg and traditional regressors. The left half of each paired column displays the training R^2 score, while the right half showcases the corresponding test R^2 score.

METHOD	$\mathbb{A}^1$		$\mathbb{B}^1$		$\mathbb{A}^2$		$\mathbb{B}^2$	
OLS Linear	0.037	0.042	0.963	0.951	0.085	0.092	0.984	0.984
Theil-Sen	-44.7	-54.4	0.958	0.946	-0.41	-3.81	0.982	0.982
RANSAC	-1.21	-1.43	0.963	0.951	-27.0	-27.1	0.983	0.983
Huber	0.036	0.041	0.962	0.949	0.085	0.092	0.984	0.984
Ridge	0.031	0.038	0.963	0.951	0.055	0.061	0.984	0.984
RidgeCV	0.037	0.042	0.963	0.951	0.085	0.092	0.984	0.984
SGD	0.009	0.01	0.962	0.95	0.005	0.004	0.983	0.983
KRR	0.0036	0.04	0.97	0.962	0.056	0.051	0.993	0.992
SVR	0.11	0.101	0.97	0.962	0.29	0.308	0.992	0.992
Voting	0.852	0.852	0.942	0.917	0.869	0.868	0.951	0.946
LReg	1.0	1.0	0.971	0.963	0.999	0.999	0.992	0.992

We also explore training LReg from scratch on non-regression tasks. Here, we delve into the practical application of LReg for text feature extraction using the AG News dataset [19] for classification tasks. Our experimental findings reveal that LReg can directly extract features from raw text. However, recognizing that tokens are unordered categorical variables, to enhance performance, we add a preprocessing layer that converts each token to a four-dimensional vector, the proportion of the token appearing in four categories.

Our experiment showcases the LReg achieving 90.01% test accuracy after the first epoch, with 90.4% test accuracy reached within just five epochs. It possesses a unique characteristic in text classification tasks: the number of its parameters remains independent of the token count. Compared to word2vec-based networks, our model performs equally well in classification but boasts only 256 model parameters, a significant reduction from the word2vec-based network with more than 6.13 million parameters.

3.3. Improving pre-trained Model Performance and Continuously Reducing Loss

As described in Sec. 2.6, LReg can serve as a connection to enable the loss function of large pre-trained models to escape local minima (see Figure 5). In this module, the SigmoidNorm layer and linear head contain trainable parameters (see Figure 2c). In this section, we illustrate the effectiveness of LReg connection through ablation studies. To ensure a fair comparison with baseline models, we adopt a stricter experimental setup: *the original training recipe of the pre-trained model remains unchanged, the entire pre-trained model is frozen, and no additional training data is used*.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 June 2025 doi:10.20944/preprints202506.1335.v1

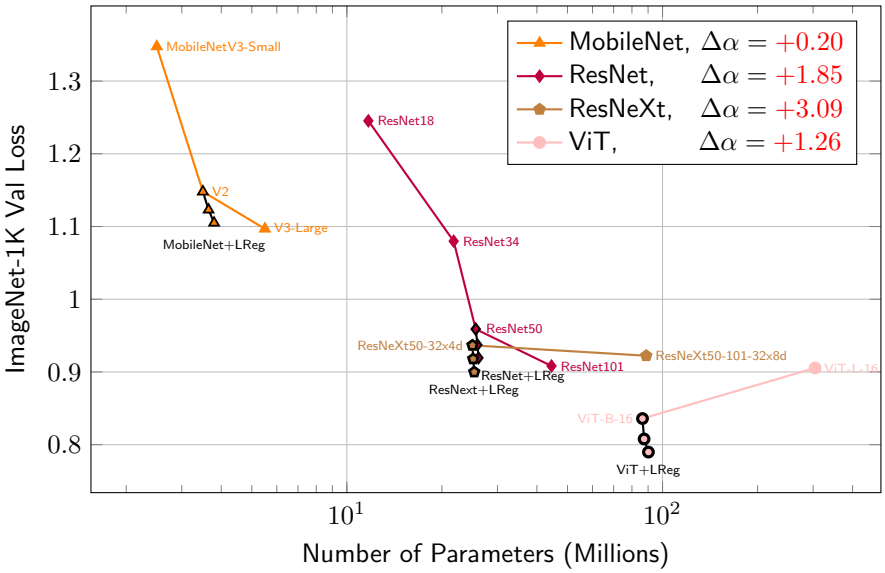

Figure 5. LReg connection greatly reduces the loss function value. Colored lines represent pre-trained models across varying scales, while black lines represent our method, which significantly decreases cross-entropy loss on the ImageNet-1 validation set and outperforms model scaling with negligible changes to the model size.

Table 2. Comparison of LReg with baseline models across different architectures. LReg improves top-1 accuracy while introducing only a negligible increase in trainable parameters. It also achieves either comparable or superior training efficiency by converging within five training epochs, thereby significantly reducing training time compared to full model training. Unlike baseline models that require days of training, LReg enhances performance within an hour. These results demonstrate the effectiveness of LReg as a general, parameter-efficient fine-tuning method for large-scale models, without being limited to transformer-based architectures. * indicates numbers published in prior works.

Model	Method	# Trainable Parameters	Acc@1	Acc@5	Speed (img/s)	Training Time (total)
MobileNet-V2	Baseline*	3.5 M	71.878	**90.286**	701.51	16h 37m
	LReg	328,720	**71.934**	90.268	685.17	**32m 12s**
ResNet-50	Baseline*	25.6 M	76.130	92.862	440.18	2d 1h 15m
	LReg	540,688	**76.274**	**92.932**	411.50	**40m 42s**
ResNeXt-50_32x4d	Baseline*	25.0 M	77.618	**93.698**	334.72	3d 1h 32m
	LReg	135,172	**77.650**	93.672	315.33	**51m 9s**
ViT-B16	Baseline*	86.6 M	81.072	**95.318**	331.02	3d 3h 26m
	LReg	67,076	**81.082**	95.316	316.31	**56m 54s**

This restriction is crucial to ablation studies because the training recipes for pre-trained models often have room for optimization [20]. Changes to batch size, optimizer, weight decay rate, or the order of applying data augmentations could potentially improve the performance of the pre-trained model [21,22]. To avoid such effects, we selected pre-trained models from TorchVision with publicly available training recipes as our baselines. The weights of these pre-trained models closely reproduce the results from the original papers on the ImageNet-1K dataset [23], with recipes available at [24].

Our method offers significant practical value. Unlike model scaling and adaptation approaches, it requires no modifications to the raw training recipe, converges within a single epoch, and is applicable to any model, not limited to the transformer-based architecture. For instance, on 4x A6000 GPUs, it adds fewer than one million parameters to a ResNet-50 model and achieves a 0.2% accuracy gain in just 40 minutes of training. Due to space constraints, we present some of the comparison results in Table 2.

 doi:10.20944/preprints202506.1335.v1

In Appendix B, we highlight the strong stability of LReg, showing ±0.07% fluctuation in validation accuracy across independent model training (see Appendix B, Table A1). Trainable parameters in our method originate from the SigmoidNorm and linear layer (see Figure 2c). The total number of trainable parameters can be calculated as $N = (C_{\text{in}} + 1) \times d + d \times n \times C$, where N is the number of model parameters, C_{in} is the SigmoidNorm input dimension, d the PCA output dimension, n is the mesh node count (degrees of freedom), and C the total output dimension.

Increasing n and d improves model performance (see Appendix B, Table A2). We also study empirical scaling laws for model performance on cross-entropy loss to demonstrate how our method further the loss value of pre-trained models. [5] proposed an empirical formula where the loss scales as a power-law with model size:

$$L(N) = \left(\frac{c}{N}\right)^{\alpha}$$

where $L(N)$ is the cross-entropy loss on the validation dataset, c and α are constants related to the model type. Figure 5 shows that our method is more effective than simply scaling the model size. Our method increases the α for MobileNet [25] from 0.27 to 0.47, for ResNet [26] from 0.25 to 2.1, for ResNeXt [27] from 0.01 to 3.1, and for ViT [28] from -0.06 to 1.2.

3.4. Transfer Learning via LReg Connection

Through experiments, we identified three key limitations of existing PEFT methods: 1) Dependence on Transfer Learning: Methods like LoRA [29] struggle to outperform pre-trained models on the same or similar datasets, when no domain adaptation is needed. This means they cannot continuously reduce the loss function on the raw training set. Additional training data is often required for these methods to be effective. *Therefore, their strength lies mainly in transfer learning scenarios* (see Appendix B); 2) Sensitive Training Requirements: PEFT methods require specific training recipes, such as small learning rates. Without these, performance often deteriorates from the first epoch; 3) Task and Architecture Limits: PEFT methods, like LoRA, LoHA [30] and IA3 [31], are restricted to transformer-based architectures.

We compare fine-tuning (FT), BitFit [32], LoRA, and LReg connection on the GLUE dataset [33]; subsequently, we conduct additional experiments on WikiText benchmarks [34] using GPT2 [35] for causal language modeling and Roberta [36] for masked language modeling with a default random seed (42). Pre-trained models of the HuggingFace Transformers library [37] are used. A reduced learning rate ($\leq 10^{-5}$) is applied for LoRA to prevent performance degradation in these experiments. As shown in Table 3, our method overcomes the mentioned challenges, performing well even without additional training data. Therefore, it is also versatile for transfer learning.

Table 3. GPT2 and RoBERTa with Different Adaptation Methods on the GLUE dataset and WikiText Benchmark. We report the overall (both matched and mismatched) accuracy for MNLI and the accuracy for other tasks. For each model, we conducted multiple experiments and show the accuracy as "mean$_{\pm\text{std}}$". For a fair comparison, we adjusted the number of parameters of our method to be similar to those of LoRA. * indicates numbers published in prior works. We also report the number of trainable parameters, perplexity (PPL, with lower values being preferable), and training throughput (measured in sequences per second) for language modeling tasks.

	MNLI		SST-2		MRPC		QNLI		QQP		RTE	
Method	#Params	Accuracy	#Params	Accuracy	#Params	Accuracy	#Params	Accuracy	#Params	Accuracy	#Params	Accuracy
FT*	125M	87.6	125M	94.8	125M	90.2	125M	92.8	125M	91.9	125M	78.7
BitFit*	0.1M	84.7	0.1M	93.7	0.1M	92.7	0.1M	91.8	0.1M	84.0	0.1M	81.5
LoRA*	0.295M	$87.5_{\pm.3}$	0.295M	$95.1_{\pm.2}$	0.295M	$89.7_{\pm.7}$	0.295M	$93.3_{\pm.3}$	0.295M	$90.8_{\pm.1}$	0.295M	$86.6_{\pm.7}$
LReg	0.259M	$87.9_{\pm.5}$	0.259M	$95.3_{\pm.6}$	0.259M	$90.5_{\pm.4}$	0.259M	$93.5_{\pm.2}$	0.259M	$89.7_{\pm.4}$	0.259M	$83.5_{\pm.3}$

	GPT2			GPT2-Medium			GPT2-Large			Roberta-Base		
Method	#Params	PPL	Seq/s	#Params	PPL	Seq/s	#Params	PPL	Seq/s	#Params	PPL	Seq/s
FT	124.4M	21.4234	6.98	354.8M	15.8900	3.50	774.0M	13.8468	1.58	124.7M	3.6415	10.05
LoRA	0.295M	21.4188	**12.90**	0.393M	15.8873	6.29	0.737M	13.8486	1.67	0.295M	3.6425	9.85
LReg	0.203M	**21.3401**	9.58	0.404M	**15.8674**	**6.66**	0.607M	**13.8443**	**2.52**	0.303M	**3.6414**	**12.37**

 doi:10.20944/preprints202506.1335.v1

4. Discussion

It is important to acknowledge the limitations associated with LReg. As described in Sec. 2.6, the substantial computational cost of extracting features from high-dimensional data restricts its direct application to large-scale datasets. Presently, we address this by incorporating LReg as an adaptation method, but we aim to extract features directly from large-scale data in the future. Additionally, the experiments in Sec. 3.3 indicate that, although our adaptation method significantly reduces cross-entropy on the ImageNet-1K validation set, the extent of accuracy improvement is relatively modest, warranting further investigation. We didn't compare LReg with models such as Support Vector Machine (SVM) known for their robust interpretability, due to fundamentally different underlying principles, and the usual missing experimental results from interpretable models on large datasets.

5. Conclusions

In this paper, we reformulate MLPs and introduce a mathematically equivalent framework named LReg. Furthermore, we propose a discrete method, referred to as the MRC loop, which serves as an alternative to traditional continuous optimization techniques, thereby enabling the loss function to reach global minima. For general pre-trained models, we demonstrate that our approach can significantly reduce the loss function value.

From a theoretical perspective, we construct a visual example to elucidate the intrinsic challenge encountered by traditional optimization methods in reducing the loss function from local minima to global minima, and highlight the benefits of **discrete parameter updates** in addressing this challenge. Furthermore, we analyze the error-bound formula and, for the first time, derive an **explicit scaling law**, while we empirically validate through regression tasks. From an application perspective, LReg effectively mitigates the core limitation of traditional neural networks in learning multi-frequency data. Additionally, LReg serves as a plug-and-play module, and our analysis of the scaling law shows that it substantially reduces the loss function value of pre-trained models while providing moderate improvement in performance. In contrast to scaling up a model and training it from scratch, our approach is more efficient. Furthermore, LReg broadens the application scope of PEFT methods in transfer learning, without requiring the pre-trained model to be transformer-based.

This work opens new directions for research in model training and establishes a foundation for further advancements in discrete optimization.

Abbreviations

The following abbreviations are used in this manuscript:

FEM	Finite Element Method
LReg	Lagrange Regressor
MAE	Mean Absolute Error
MLP	Multi-Layer Perceptron
MRC	Mesh-Refinement-Coarsening
PCA	Principal Component Analysis
PEFT	Parameter-Efficient Fine-Tuning
SVM	Support Vector Machine

Appendix A Mesh and Lagrange Basis Function

Appendix A.1 Triangulated Mesh

In the context of FEM, elements often serve as the fundamental building blocks of the triangulation *mesh*, taking the form of *simplices* created by connecting *nodes*. For instance, in 1D FEM, simplices are intervals (see Figure 1a); in 2D FEM, triangles with three nodes are commonly used (see Figure A1, right), while 3D FEM often employs tetrahedra with four nodes. This concept is visually depicted in Figure A1 (left), where the mesh consists of eight nodes and seven triangles. This type of mesh is established by specifying the coordinates of discrete nodes and the vertex indices of simplices. Let d

 doi:10.20944/preprints202506.1335.v1

represent the dimensions, $\{\boldsymbol{p}^{(i)}\}_{i=0}^{n-1}$ denote the grid nodes, and introduce a matrix $\boldsymbol{P}$ to store the node coordinates:

$$\boldsymbol{P}_{i,j} = \boldsymbol{p}_j^{(i)}.$$

Additionally, utilize a matrix $\boldsymbol{T}$ to store the indices of nodes constituting the simplices within the triangulation. Specifically, access the j-th sorted vertex of the i-th simplex in this mesh as $\boldsymbol{P}_{\boldsymbol{T}_{i,j},:}$. Figure A1 (left) illustrate a matrix $\boldsymbol{T}$ takes the following form:

$$\boldsymbol{T} = \begin{bmatrix} 6 & 0 & 3 & 2 & 7 & 3 & 4 \\ 7 & 6 & 7 & 3 & 4 & 4 & 3 \\ 5 & 5 & 1 & 1 & 5 & 7 & 2 \end{bmatrix}^T.$$

This matrix serves to describe all seven simplices within the mesh, such as the first simplex $\triangle \boldsymbol{p}^{(6)}\boldsymbol{p}^{(7)}\boldsymbol{p}^{(5)}$ and the last simplex $\triangle \boldsymbol{p}^{(4)}\boldsymbol{p}^{(3)}\boldsymbol{p}^{(2)}$.

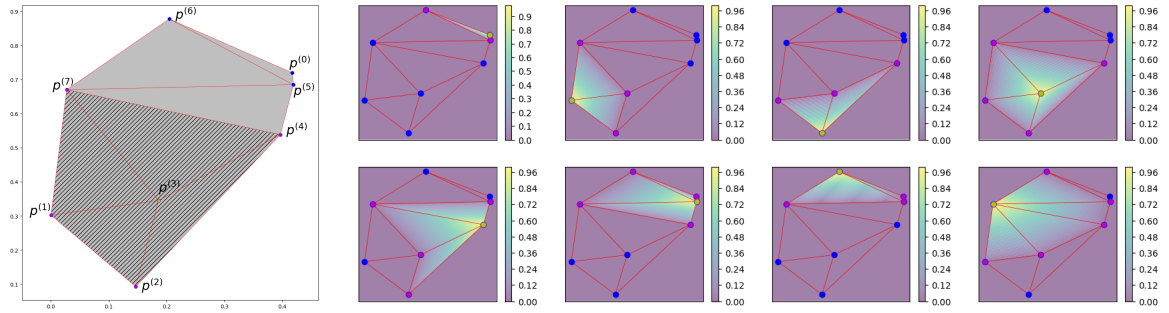

Figure A1. Left - Mesh with eight nodes and seven triangles. Right - Contours of eight Lagrange basis functions, linear variation of ψ_i associated with node $\boldsymbol{p}^{(i)}$ across all triangles.

The first-order Lagrange basis studied in this article, denoted as $\{\psi_0(\boldsymbol{x}), \cdots, \psi_{n-1}(\boldsymbol{x})\} \subset P_1(\mathbb{R}^d)$, are piecewise linear polynomials associated with nodes $\{\boldsymbol{p}^{(0)}, \cdots, \boldsymbol{p}^{(n-1)}\}$. These functions are defined such that $\psi_i(\boldsymbol{p}^{(j)}) = \mathbb{1}_{\mathrm{i=j}}$. Figure A1 (right) illustrates this: $\psi_i(\boldsymbol{x})$ corresponds to node $\boldsymbol{p}^{(i)}$, exhibiting linear variation across all elements. Its support encompasses the union of all neighboring elements of node $\boldsymbol{p}^{(i)}$ (refer to Appendix B for a 3-dimensional visualization). For example, $\mathrm{supp}(\psi_3) = \triangle \boldsymbol{p}^{(3)}\boldsymbol{p}^{(4)}\boldsymbol{p}^{(7)} \cup \triangle \boldsymbol{p}^{(3)}\boldsymbol{p}^{(7)}\boldsymbol{p}^{(1)} \cup \triangle \boldsymbol{p}^{(3)}\boldsymbol{p}^{(1)}\boldsymbol{p}^{(2)} \cup \triangle \boldsymbol{p}^{(3)}\boldsymbol{p}^{(2)}\boldsymbol{p}^{(4)}$.

Now, we formulate the Lagrange basis from its original definition to establish the foundational architecture of LReg. It is important to highlight that the traditional Lagrange basis involves unbalanced computing of barycentric coordinates, which may not be well-suited for parallel deep learning platforms (see Appendix C for details on the traditional definition of the Lagrange basis). Consequently, in this subsection, we re-derive the Lagrange basis to enhance parallel computing.

Appendix A.2 Lagrange Basis Function

Let n_t represent the number of simplices in the multiscale mesh. We introduce the Parameters Tensor $\mathbf{S}$ defined as:

$$\mathbf{S}_{j,:,:} = \begin{bmatrix} \boldsymbol{p}_0^{(\boldsymbol{T}_{j,0})} & \cdots & \boldsymbol{p}_{d-1}^{(\boldsymbol{T}_{j,0})} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ \boldsymbol{p}_0^{(\boldsymbol{T}_{j,d-1})} & \cdots & \boldsymbol{p}_{d-1}^{(\boldsymbol{T}_{j,d-1})} & 1 \\ \boldsymbol{p}_0^{(\boldsymbol{T}_{j,d})} & \cdots & \boldsymbol{p}_{d-1}^{(\boldsymbol{T}_{j,d})} & 1 \end{bmatrix}^{-1}, \quad j = 0, \cdots, n_t - 1.$$

 doi:10.20944/preprints202506.1335.v1

Additionally, we introduce the Node Membership tensor $\mathbf{M}$ defined as:

$$\mathbf{M}_{i,j,k} = \begin{cases} 1, & \text{if the } i\text{-th node matches the } k\text{-th vertex of the } j\text{-th simplex,} \\ 0, & \text{other cases.} \end{cases}$$

By defining:

$$\boldsymbol{U}_{j,k}(\boldsymbol{x}) = \sum_{\tau=0}^{d-1} \mathbf{S}_{j,\tau,k} \cdot \boldsymbol{x}_\tau + \mathbf{S}_{j,d,k}, \quad j = 0, \cdots, n_t - 1,\ k = 0, \cdots, d.$$

We will demonstrate in Appendix A that the following function qualifies the definition of Lagrange basis:

$$\psi_i(\boldsymbol{x}) = \frac{\sum_{j=0}^{n_t-1} \sum_{k=0}^{d} \mathbb{1}_{\min_\tau \boldsymbol{U}_{j,\tau}(\boldsymbol{x}) \geq 0} \cdot \mathbf{M}_{i,j,k} \cdot \boldsymbol{U}_{j,k}(\boldsymbol{x})}{\max(\sum_{j=0}^{n_t-1} \sum_{k=0}^{d} \mathbb{1}_{\min_\tau \boldsymbol{U}_{j,\tau}(\boldsymbol{x}) \geq 0} \cdot \mathbf{M}_{i,j,k}, 1)}, \quad i = 0, \cdots, n-1. \tag{A1}$$

So far, we have successfully constructed :

$$\begin{aligned} \text{Encoder} : &\mathbb{R}^d \to [0,1]^n, \\ &\boldsymbol{x} \mapsto (\psi_0(\boldsymbol{x}), \cdots, \psi_{n-1}(\boldsymbol{x})). \end{aligned}$$

The above basis is in the format of $P_1(\mathbb{R}^d)$, which is very useful for low-dimensional regression tasks. In the one-dimensional case, Eq. (A1) simplifies to the Lagrange basis $P_1(\mathbb{R}^1)$:

$$\psi_i(\boldsymbol{x}) = \min\left(\frac{\text{ReLU}(\boldsymbol{x} - \boldsymbol{p}_{i-1})}{\boldsymbol{p}_i - \boldsymbol{p}_{i-1}}, \frac{\text{ReLU}(\boldsymbol{p}_{i+1} - \boldsymbol{x})}{\boldsymbol{p}_{i+1} - \boldsymbol{p}_i}\right), \quad i = 0, 1, \cdots, n-1.$$

The $P_1(\mathbb{R}^1)$ basis has exceptionally low computational complexity, making its direct sum well-suited for recognition tasks on large-scale datasets. In Sec. 2.6, we introduce the specific approach in detail. We use the $P_1(\mathbb{R}^1)$ basis to implement an adaptive method for learning a decoupled residual system. This method can enhance the performance of pre-trained models on large datasets.

Appendix A.3 Proof of Lagrange Basis Expression

We will now demonstrate, in three concise steps, that Eqn. (A1) qualifies as a Lagrange basis function.

Piecewise Linear: Since $\{\boldsymbol{U}_{j,0}, \cdots, \boldsymbol{U}_{j,d}\}$ are piecewise linear functions, their linear combination L_i is also piecewise linear.

Kronecker Delta: From the definition of $\boldsymbol{U}$ and $\mathbf{S}$, we have the following equation:

$$\begin{bmatrix} \boldsymbol{x}_0 & \cdots & \boldsymbol{x}_{d-1} & 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{U}_{j,0}(\boldsymbol{x}) & \cdots & \boldsymbol{U}_{j,d}(\boldsymbol{x}) \end{bmatrix} \begin{bmatrix} \boldsymbol{p}_0^{(\boldsymbol{T}_{j,0})} & \cdots & \boldsymbol{p}_{d-1}^{(\boldsymbol{T}_{j,0})} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ \boldsymbol{p}_0^{(\boldsymbol{T}_{j,d-1})} & \cdots & \boldsymbol{p}_{d-1}^{(\boldsymbol{T}_{j,d-1})} & 1 \\ \boldsymbol{p}_0^{(\boldsymbol{T}_{j,d})} & \cdots & \boldsymbol{p}_{d-1}^{(\boldsymbol{T}_{j,d})} & 1 \end{bmatrix}.$$

Decomposing this equation, we obtain $\boldsymbol{x} = \sum_{k=0}^{d} \boldsymbol{U}_{j,k}(\boldsymbol{x}) \boldsymbol{p}^{(\boldsymbol{T}_{j,k})}$ and $\sum_{k=0}^{d} \boldsymbol{U}_{j,k}(\boldsymbol{x}) = 1$. This implies two important conclusions: $\boldsymbol{U}_{j,k'}(\boldsymbol{p}^{(\boldsymbol{T}_{j,k''})}) = \mathbb{1}_{k'=k''}$ and $\min_\tau \boldsymbol{U}_{j,\tau}(\boldsymbol{x}) \geq 0$ is true if and only if $\boldsymbol{x}$ belongs to the j-th simplex. Therefore, we have

$$\begin{cases} \mathbb{1}_{\min_\tau \boldsymbol{U}_{j,\tau}(\boldsymbol{p}^{(i)}) \geq 0} = \mathbf{M}_{i,j,k} = \boldsymbol{U}_{j,k}(\boldsymbol{p}^{(i)}) = 1, & \text{if } i = \boldsymbol{T}_{j,k} \\ \mathbf{M}_{i,j,k} = \boldsymbol{U}_{j,k}(\boldsymbol{p}^{(i)}) = 0, & \text{if } i \neq \boldsymbol{T}_{j,k} \end{cases}$$

 doi:10.20944/preprints202506.1335.v1

This proves that $\psi_i(\boldsymbol{p}^{(j)}) = \mathbb{1}_{i=j}$.
Globally Continuity: Lastly, since ψ_i is inherently linear within all simplices and exhibits continuity across all grid nodes, we can conclude that ψ_i is globally continuous.

Appendix A.4 Validation of Lagrange Basis Functions' Mathematical Expression

This section focuses on the mathematical validation of Lagrange basis function expressions to ensure they satisfy our definition in Sec. 2.3, *the Lagrange basis functions are defined as tent-shaped linear functions that take a value of 1 at their specified nodes and 0 at all other nodes.* We start with a two-dimensional case and a triangulated mesh. Let triangle $\triangle$ be defined by nodes $\{\boldsymbol{p}^{(i)}, \boldsymbol{p}^{(j)}, \boldsymbol{p}^{(k)}\}$. The following barycentric coordinates $\{\lambda_{\triangle,i}, \lambda_{\triangle,j}, \lambda_{\triangle,k}\}$ are three first-degree polynomials of $\boldsymbol{x}$

$$\begin{bmatrix} \lambda_{\triangle,i}(\boldsymbol{x}) \\ \lambda_{\triangle,j}(\boldsymbol{x}) \\ \lambda_{\triangle,k}(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} \boldsymbol{p}_0^{(i)} & \boldsymbol{p}_0^{(j)} & \boldsymbol{p}_0^{(k)} \\ \boldsymbol{p}_1^{(i)} & \boldsymbol{p}_1^{(j)} & \boldsymbol{p}_1^{(k)} \\ 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_1 \\ 1 \end{bmatrix}.$$

Referring to the instance depicted in Figure A1 (left), the mesh consists of eight nodes and seven triangles. Specifically, let $\{\triangle^{(j)} = \triangle \boldsymbol{p}^{(\boldsymbol{T}_{j,0})} \boldsymbol{p}^{(\boldsymbol{T}_{j,1})} \boldsymbol{p}^{(\boldsymbol{T}_{j,2})} | j = 0, \cdots, 6\}$. We will now verify that the following ψ_3 corresponds to the third basis function in this mesh

$$\psi_3(\boldsymbol{x}) = \frac{1}{\max(\sum_{i\in\{2,3,6,5\}} \mathbb{1}_{\boldsymbol{x}\in\triangle^{(i)}}, 1)} \sum_{i\in\{2,3,6,5\}} \mathbb{1}_{\boldsymbol{x}\in\triangle^{(i)}} \lambda_{\triangle^{(i)},3}(\boldsymbol{x}).$$

First, ψ_3 possesses values of *Kronecker Delta*:

$$\psi_3(\boldsymbol{x}) = \begin{cases} \frac{1}{\max(0,1)}(0\cdot\lambda_{\triangle^{(2)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(3)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(6)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(5)},3}(\boldsymbol{x}))=0, \text{ if } \boldsymbol{x}=\boldsymbol{p}^{(0)}, \\ \frac{1}{\max(2,1)}(1\cdot 0 \quad +1\cdot 0 \quad +0\cdot\lambda_{\triangle^{(6)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(5)},3}(\boldsymbol{x}))=0, \text{ if } \boldsymbol{x}=\boldsymbol{p}^{(1)}, \\ \frac{1}{\max(2,1)}(0\cdot\lambda_{\triangle^{(2)},3}(\boldsymbol{x})+1\cdot 0 \quad +1\cdot 0 \quad +0\cdot\lambda_{\triangle^{(5)},3}(\boldsymbol{x}))=0, \text{ if } \boldsymbol{x}=\boldsymbol{p}^{(2)}, \\ \frac{1}{\max(4,0)}(1\cdot 1 \quad +1\cdot 1 \quad +1\cdot 1 \quad +1\cdot 1) \quad =1, \text{ if } \boldsymbol{x}=\boldsymbol{p}^{(3)}, \\ \frac{1}{\max(2,1)}(0\cdot\lambda_{\triangle^{(2)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(3)},3}(\boldsymbol{x})+1\cdot 0 \quad +1\cdot 0) \quad =0, \text{ if } \boldsymbol{x}=\boldsymbol{p}^{(4)}, \\ \frac{1}{\max(0,1)}(0\cdot\lambda_{\triangle^{(2)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(3)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(6)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(5)},3}(\boldsymbol{x}))=0, \text{ if } \boldsymbol{x}=\boldsymbol{p}^{(5)}, \\ \frac{1}{\max(0,1)}(0\cdot\lambda_{\triangle^{(2)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(3)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(6)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(5)},3}(\boldsymbol{x}))=0, \text{ if } \boldsymbol{x}=\boldsymbol{p}^{(6)}, \\ \frac{1}{\max(2,1)}(1\cdot 0 \quad +0\cdot\lambda_{\triangle^{(3)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(6)},3}(\boldsymbol{x})+1\cdot 0) \quad =0, \text{ if } \boldsymbol{x}=\boldsymbol{p}^{(7)}, \end{cases}$$

 doi:10.20944/preprints202506.1335.v1

Then, ψ_3 is a first-degree polynomial in every triangle and supp $\psi_3 = \text{inn} \cup_{i\in\{2,3,6,5\}} \triangle^{(i)}$:

$$\psi_3(\boldsymbol{x}) = \begin{cases} \frac{1}{\max(0,1)}(0\cdot\lambda_{\triangle^{(2)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(3)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(6)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(5)},3}(\boldsymbol{x}))=0, & \text{if } \boldsymbol{x}\in\text{inn } \boldsymbol{T}_0, \\ \frac{1}{\max(0,1)}(0\cdot\lambda_{\triangle^{(2)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(3)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(6)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(5)},3}(\boldsymbol{x}))=0, & \text{if } \boldsymbol{x}\in\text{inn } \boldsymbol{T}_1, \\ \frac{1}{\max(1,1)}(1\cdot\lambda_{\triangle^{(2)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(3)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(6)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(5)},3}(\boldsymbol{x}))=\lambda_{\triangle^{(2)},3}(\boldsymbol{x}), & \text{if } \boldsymbol{x}\in\text{inn } \boldsymbol{T}_2, \\ \frac{1}{\max(1,1)}(0\cdot\lambda_{\triangle^{(2)},3}(\boldsymbol{x})+1\cdot\lambda_{\triangle^{(3)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(6)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(5)},3}(\boldsymbol{x}))=\lambda_{\triangle^{(3)},3}(\boldsymbol{x}), & \text{if } \boldsymbol{x}\in\text{inn } \boldsymbol{T}_3, \\ \frac{1}{\max(0,1)}(0\cdot\lambda_{\triangle^{(2)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(3)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(6)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(5)},3}(\boldsymbol{x}))=0, & \text{if } \boldsymbol{x}\in\text{inn } \boldsymbol{T}_4, \\ \frac{1}{\max(1,1)}(0\cdot\lambda_{\triangle^{(2)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(3)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(6)},3}(\boldsymbol{x})+1\cdot\lambda_{\triangle^{(5)},3}(\boldsymbol{x}))=\lambda_{\triangle^{(5)},3}(\boldsymbol{x}), & \text{if } \boldsymbol{x}\in\text{inn } \boldsymbol{T}_5, \\ \frac{1}{\max(1,1)}(0\cdot\lambda_{\triangle^{(2)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(3)},3}(\boldsymbol{x})+1\cdot\lambda_{\triangle^{(6)},3}(\boldsymbol{x})+0\cdot\lambda_{\triangle^{(5)},3}(\boldsymbol{x}))=\lambda_{\triangle^{(6)},3}(\boldsymbol{x}), & \text{if } \boldsymbol{x}\in\text{inn } \boldsymbol{T}_6, \end{cases}$$

Finally, since ψ_3 is continuous in all nodes and first-degree in all triangles, it is globally continuous.

Appendix A.5 Visualization of Lagrangian basis

In Sec. 2.3 and Appendix A.2, we introduced first-order Lagrange basis functions, a set of piecewise linear functions defined on a mesh. Each basis function corresponds to a node.

Consider the grid depicted in Figure A2 (left). Taking the inserted fine node $\boldsymbol{p}^{(20)}$ as an example, it has a total of four neighboring nodes: $\boldsymbol{p}^{(5)}$, $\boldsymbol{p}^{(0)}$, $\boldsymbol{p}^{(7)}$, and $\boldsymbol{p}^{(4)}$. By connecting these nodes, we can determine the support of the basis function ψ_{20}. In Figure A2 (middle), we present the function graphs of ψ_{20} and ψ_7. It can be observed that these functions exhibit linear variations on each mesh triangle. Taking ψ_{20} as an example, its function value at $\boldsymbol{p}_{20}$ is 1, and 0 at all other nodes. Similarly, ψ_7 has a function value of 1 at $\boldsymbol{p}_7$ and 0 at other nodes. In Figure A2 (right), the orange triangles represent the function graph of $f(\boldsymbol{x})$ on the domain $\triangle\boldsymbol{p}^{(0)}\boldsymbol{p}^{(7)}\boldsymbol{p}^{(9)}$, the green dots represent a subset of the training set $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_i$, where the projections (raw data) fall on $\triangle\boldsymbol{p}^{(0)}\boldsymbol{p}^{(7)}\boldsymbol{p}^{(9)}$. Base on the definition of Lagrange basis function, we have:

$$f(\boldsymbol{x})|_{\boldsymbol{x}\in\triangle\boldsymbol{p}^{(0)}\boldsymbol{p}^{(7)}\boldsymbol{p}^{(9)}} = \sum_i c_i\psi_i(\boldsymbol{x})|_{\boldsymbol{x}\in\triangle\boldsymbol{p}^{(0)}\boldsymbol{p}^{(7)}\boldsymbol{p}^{(9)}} = c_0\psi_0(\boldsymbol{x}) + c_7\psi_7(\boldsymbol{x}) + c_9\psi_9(\boldsymbol{x}).$$

The **training process** mentioned in Sec. 2.4 uses the least squares method to adjust coefficients, minimizing the mean gap (MAE) between the green points and the triangular surface.

$$\begin{cases} \arg\min_{(c_0,c_7,c_9)} \sum_i |c_0\psi_0(\boldsymbol{x}^{(i)}) + c_7\psi_0(\boldsymbol{x}^{(i)}) + c_9\psi_0(\boldsymbol{x}^{(i)}) - y^{(i)}|, & \text{Using model training method;} \\ \arg\min_{(c_0,c_7,c_9)} \sum_i |c_0\psi_0(\boldsymbol{x}^{(i)}) + c_7\psi_0(\boldsymbol{x}^{(i)}) + c_9\psi_0(\boldsymbol{x}^{(i)}) - y^{(i)}|^2, & \text{Using least squares method.} \end{cases}$$

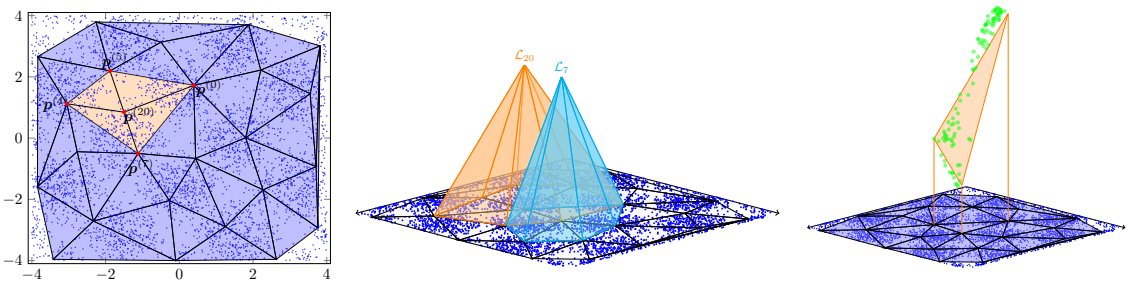

Figure A2. Data visualization. Left - an example of refining a two-dimensional mesh. Middle - the graphs of basis functions ψ_{20} and ψ_7. Right - the graphs of the function $f(\boldsymbol{x})$ and a subset of the training set, adjusting coefficients for minimizing MAE Loss.

 doi:10.20944/preprints202506.1335.v1

Appendix B Additional Experiments

Table A1. Classification accuracy on the ImageNet-1K validation set and one epoch training, with all methods adhering to the raw training recipes. We report the changes in the number of model parameters and their ratio of changes and total. For each model, we conducted multiple experiments and show the accuracy as "$\text{mean}_{\pm\text{std}}(\text{mean} - \text{baseline})$".

Model	d	n	# Params	Acc@1 (%)	Acc@5 (%)
MobileNet-V2	32	8	0.298 M (+8.51%)	$71.920_{\pm.016}(+.045)$	$90.302_{\pm.021}(+.016)$
ResNet50	32	8	0.324 M (+1.27%)	$76.334_{\pm.054}(+.204)$	$92.976_{\pm.024}(+.114)$
ResNeXt50	32	8	0.324 M (+1.30%)	$77.796_{\pm.063}(+.178)$	$93.653_{\pm.038}(+.178)$
ViT-B-16	8	32	1.032 M (+1.19%)	$81.092_{\pm.012}(+.020)$	$95.316_{\pm.004}(-.002)$

We study the robustness of the LReg on the ImageNet-1K validation set with one training epoch. The result is presented in Table A1. All methods follow the raw training recipes to ensure fair comparisons.

Increasing n, d, and training epochs is an effective way to enhance model performance, as shown in Table A2. Unlike the function fitting task, diminishing returns in performance improvements are observed in this case due to the finite cardinality of image and text datasets.

Table A2. ResNet-50 Performance with different n and d. The table explores the impact of varying n (PCA output dimension) and d (degrees of freedom) on the performance of the LagEncoder-based ResNet-50 model. Metrics include the number of additional parameters, top-1 (Acc@1) and top-5 (Acc@5) validation accuracy, and training speed (images per second). The baseline ResNet-50 has 25.6M parameters, Acc@1: 76.130%, and Acc@5: 92.862%.

Coeff (n, d)	(4, 4)	(4, 8)	(4, 16)	(4, 32)	(8, 4)	(8, 8)	(8, 16)	(8, 32)
# Parameters	36,868	69,636	135,172	266,244	73,736	139,272	270,344	532,488
Acc@1	76.226	76.226	**76.242**	76.222	76.238	76.256	76.268	**76.276**
Acc@5	92.952	92.960	92.948	**92.972**	92.964	92.954	**92.966**	92.952
Speed (img/s)	405.31	411.18	390.26	**418.84**	382.22	394.63	**409.52**	404.43
Coeff (n, d)	(16, 4)	(16, 8)	(16, 16)	(16, 32)	(32, 4)	(32, 8)	(32, 16)	(32, 32)
# Parameters	147,472	278,544	540,688	1,064,976	294,944	557,088	1,081,376	2,129,952
Acc@1	76.272	76.244	76.248	**76.284**	**76.298**	76.272	76.248	76.294
Acc@5	92.940	92.946	**92.968**	92.958	92.956	92.958	**92.968**	92.932
Speed (img/s)	393.19	**394.86**	379.72	380.30	393.34	**414.39**	379.72	368.94

Appendix C Additional Applications

Appendix C.1 Solve PDEs

In this section, we utilize the LReg to address the following partial differential equations (PDEs):

$$\begin{cases} \Delta u + (u - \beta)^2 = (\alpha \cos x \sin y - 1)^2 + 1, & (x, y) \in \Omega; \\ u = \beta, & (x, y) \in \partial\Omega. \end{cases}$$

Here, $\Omega = [0, 1] \times [0, 1]$. We construct a dataset that takes (α, β) as input data and assigns the corresponding numerical solution of the PDEs as the target output. This dataset comprises 12,000 examples, with α randomly selected from the distribution $U(-\pi/2, \pi/2)$ and β randomly chosen from the distribution $U(0, 2\pi)$. We then split the dataset into two parts: 10,000 for training and 2,000 for testing. Figure A3 illustrates how well the network predicts the exact solution.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 June 2025 doi:10.20944/preprints202506.1335.v1

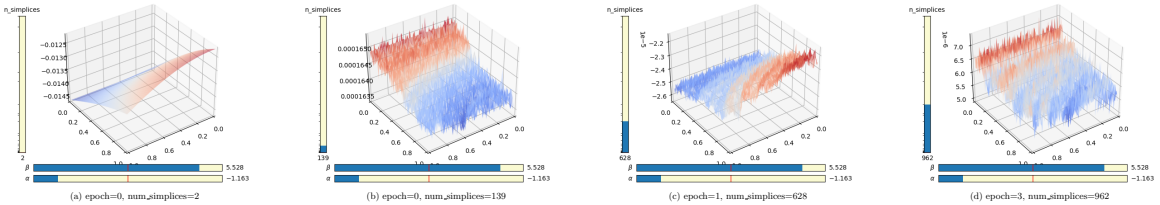

Figure A3. Residual - The gap between the exact solution and the model output.

Appendix C.2 fitting high-noise data

In this section, we conduct the LReg on learning the dataset $\mathbb{A} = \{(x,y)|X \sim U(-4,4), Y \sim \mathcal{N}(\sin x, 0.2\cos^2 x)\}$. A comprises 6000 examples, with 5000 for training and 1000 for testing. Figure A4 shows the training progress.

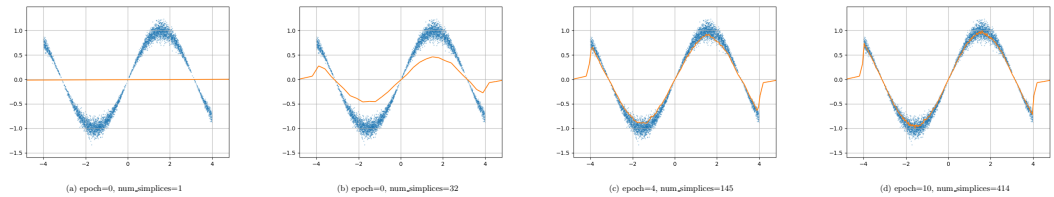

Figure A4. Blue dots represent the training set, while the orange curve represents the network.

Appendix C.3 fitting multi-frequency data

In this section, we conduct the LReg on learning the dataset $\mathbb{A} = \{(x,y)|\frac{1}{x} \sim U(0.02,0.5), y = \sin\frac{1}{x}\}$. A comprises 6000 examples, with 5000 for training and 1000 for testing. Figure A5 illustrates the training progress. Remarkably, after just 4 epochs of training, the neural network outputs closely approximate the target values. By the 32nd epoch's conclusion, the neural network outputs and target values are nearly indistinguishable.

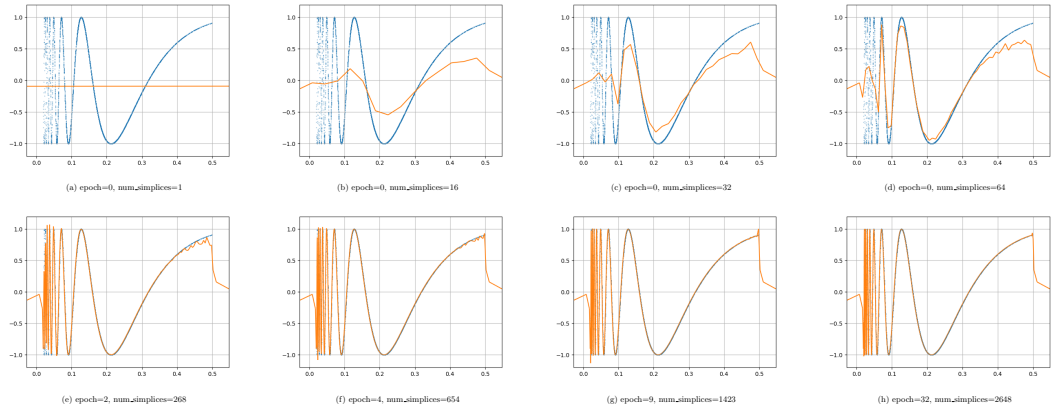

Figure A5. Blue dots represent the training set, while the orange curve represents the network.

Appendix C.4 Fit a Vector-Valued Function

In this instance, we utilize the LReg to fit spherical harmonics. Our dataset denoted as $\mathbb{A} = \{(\boldsymbol{x},\boldsymbol{y})|\boldsymbol{x} = (\theta,\phi), \boldsymbol{y} = (\text{Real}(Y_4^2(\theta,\phi)), \text{Imag}(Y_4^2(\theta,\phi))), \Theta \sim U(0,2\pi), \Phi \sim U(0,\pi)\}$, comprises 48,000 examples, with 40,000 allocated for training and an additional 8,000 for testing. Figure A6 shows the training progress.

 doi:10.20944/preprints202506.1335.v1

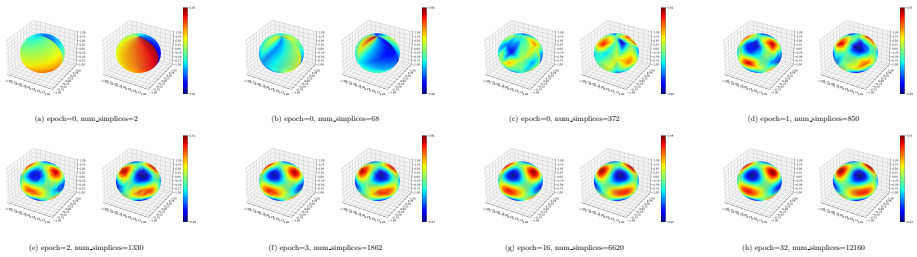

Figure A6. In each block, the left panel represents the real part of our model output, while the right panel represents the imaginary part of the model output.

References

1. Saxe, A.M.; McClelland, J.L.; Ganguli, S. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120* **2013**.
2. Dauphin, Y.N.; Pascanu, R.; Gulcehre, C.; Cho, K.; Ganguli, S.; Bengio, Y. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *Advances in neural information processing systems* **2014**, *27*.
3. Goodfellow, I.J.; Vinyals, O.; Saxe, A.M. Qualitatively characterizing neural network optimization problems. *arXiv preprint arXiv:1412.6544* **2014**.
4. Choromanska, A.; Henaff, M.; Mathieu, M.; Arous, G.B.; LeCun, Y. The loss surfaces of multilayer networks. In Proceedings of the Artificial intelligence and statistics. PMLR, 2015, pp. 192–204.
5. Kaplan, J.; McCandlish, S.; Henighan, T.; Brown, T.B.; Chess, B.; Child, R.; Gray, S.; Radford, A.; Wu, J.; Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361* **2020**.
6. Henighan, T.; Kaplan, J.; Katz, M.; Chen, M.; Hesse, C.; Jackson, J.; Jun, H.; Brown, T.B.; Dhariwal, P.; Gray, S.; et al. Scaling laws for autoregressive generative modeling. *arXiv preprint arXiv:2010.14701* **2020**.
7. Zienkiewicz, O.C.; Taylor, R.L.; Nithiarasu, P.; Zhu, J. *The finite element method*; Vol. 3, Elsevier, 1977.
8. Xu, Z.Q.J.; Zhang, Y.; Xiao, Y. Training behavior of deep neural network in frequency domain. In Proceedings of the International Conference on Neural Information Processing. Springer, 2019, pp. 264–274.
9. Platt, J.; et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers* **1999**, *10*, 61–74.
10. Thiel, H. A rank-invariant method of linear and polynomial regression analysis. I, II, III. *Nederl. Akad. Wetensch., Proc. 53* **1950**, pp. 386–392.
11. Cantzler, H. Random sample consensus (ransac). *Institute for Perception, Action and Behaviour, Division of Informatics, University of Edinburgh* **1981**.
12. Zhang, T. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In Proceedings of the Proceedings of the twenty-first international conference on Machine learning, 2004, p. 116.
13. Hilt, D.E.; Seegrist, D.W. *Ridge, a computer program for calculating ridge regression estimates*; Vol. 236, Department of Agriculture, Forest Service, Northeastern Forest Experiment . . . , 1977.
14. Stone, M. Cross-validatory choice and assessment of statistical predictions. *Journal of the royal statistical society: Series B (Methodological)* **1974**, *36*, 111–133.
15. Jain, P.; Kakade, S.M.; Kidambi, R.; Netrapalli, P.; Sidford, A. Accelerating stochastic gradient descent for least squares regression. In Proceedings of the Conference On Learning Theory. PMLR, 2018, pp. 545–604.
16. Murphy, K.P. *Machine learning: a probabilistic perspective*; MIT press, 2012.
17. Friedman, J.H. Greedy function approximation: a gradient boosting machine. *Annals of statistics* **2001**, pp. 1189–1232.
18. Breiman, L. Random forests. *Machine learning* **2001**, *45*, 5–32.
19. Zhang, X.; Zhao, J.; LeCun, Y. Character-level convolutional networks for text classification. *Advances in neural information processing systems* **2015**, *28*.
20. Wightman, R.; Touvron, H.; Jégou, H. Resnet strikes back: An improved training procedure in timm. *arXiv preprint arXiv:2110.00476* **2021**.
21. Touvron, H.; Vedaldi, A.; Douze, M.; Jégou, H. Fixing the train-test resolution discrepancy. *Advances in neural information processing systems* **2019**, *32*.

 doi:10.20944/preprints202506.1335.v1

22. Touvron, H.; Cord, M.; Douze, M.; Massa, F.; Sablayrolles, A.; Jégou, H. Training data-efficient image transformers & distillation through attention. In Proceedings of the International conference on machine learning. PMLR, 2021, pp. 10347–10357.
23. Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. Imagenet large scale visual recognition challenge. *International journal of computer vision* **2015**, *115*, 211–252.
24. TorchVision Contributors. TorchVision Models. https://pytorch.org/vision/master/models.html, 2024. Accessed: 2024-08-09.
25. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* **2017**.
26. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.
27. Xie, S.; Girshick, R.; Dollár, P.; Tu, Z.; He, K. Aggregated residual transformations for deep neural networks. In Proceedings of the Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 1492–1500.
28. Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S.; et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* **2020**.
29. Hu, E.J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; Chen, W. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685* **2021**.
30. Hyeon-Woo, N.; Ye-Bin, M.; Oh, T.H. Fedpara: Low-rank hadamard product for communication-efficient federated learning. *arXiv preprint arXiv:2108.06098* **2021**.
31. Liu, H.; Tam, D.; Muqeeth, M.; Mohta, J.; Huang, T.; Bansal, M.; Raffel, C.A. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems* **2022**, *35*, 1950–1965.
32. Zaken, E.B.; Ravfogel, S.; Goldberg, Y. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199* **2021**.
33. Wang, A. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461* **2018**.
34. Merity, S.; Xiong, C.; Bradbury, J.; Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843* **2016**.
35. Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I.; et al. Language models are unsupervised multitask learners. *OpenAI blog* **2019**, *1*, 9.
36. Liu, Y. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* **2019**, *364*.
37. Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; et al. Transformers: State-of-the-art natural language processing. In Proceedings of the Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations, 2020, pp. 38–45.