Regression Descent: A Statistical Framework for Neural Network Optimization

Kamaljeet Singh[1,4], Nick Hengartner[1,*], Brian Bell[2],Hao Helen Zhang[4], James M. Hyman[3]

[1] Theoretical Biology and Biophysics Group, Los Alamos National Laboratory
[2] Advanced Research in Cyber Systems, Los Alamos National Laboratory
[3] Theoretical Division, Los Alamos National Laboratory
[4] Department of Mathematics, University of Arizona
{kamaljeetsingh, haozhang}@arizona.edu, {bwbell, mhyman, nickh}@lanl.gov

[*]Corresponding author

Abstract

We present Regression Descent (RD), a novel optimization algorithm for training deep neural networks that reformulates each gradient step as a regression problem in the span of the Jacobian. By leveraging the implicit function theorem in overparameterized settings where the number of parameters exceed observations ($p > n$), we project the optimization onto an n-dimensional subspace, enabling the use of statistical techniques and potentially improved conditioning. Our key insight is that in the overparameterized regime, meaningful parameter updates lie in the row space of the Jacobian matrix, allowing us to solve a lower-dimensional regression problem with explicit regularization control. We establish convergence guarantees for RD under standard smoothness assumptions, showing that it achieves a convergence rate of $O(1/k)$ for smooth non-convex objectives. The algorithm naturally handles the ill-conditioning common in neural network optimization through adaptive regularization and extends seamlessly to multi-output problems and mini-batch settings. Experimental results on Lorenz96, MNIST, and FMNIST demonstrate that RD achieves up to 40% faster convergence compared to SGD and Adam in terms of wall-clock time, with strong performance in the presence of activation function saturation. The computational overhead of solving $m \times m$ linear systems (where m is the batch size) is offset by improved convergence properties and GPU-efficient operations. Our work opens new avenues for understanding neural network optimization through the lens of statistical regression, providing a practical algorithm for scenarios where standard gradient methods struggle.
The code is available at GitHub.

Proceedings of the 29[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300.

1 INTRODUCTION

The remarkable success of deep neural networks has been enabled by sophisticated optimization algorithms that can navigate complex, non-convex loss landscapes with millions or even billions of parameters. Despite this empirical success, our theoretical understanding of why gradient-based methods work so well in practice remains incomplete. This gap is particularly pronounced in the modern overparameterized regime, where neural networks have far more parameters than training examples, yet still generalize well - a phenomenon that challenges traditional statistical wisdom. In practice, this remarkable behavior is inseparable from the algorithms used to fit these models. Gradient descent and its variants (such as SGD and Adam) are the dominant algorithms used to train neural networks (Ruder, 2017). Modern frameworks use automatic differentiation (Paszke et al., 2017) to evaluate gradients efficiently on random subsets of the data, called a *minibatch* to ensures the algorithm scales with increasing number of training data points n. The resulting Stochastic Gradient Descent (SGD) algorithm

(Ketkar, 2017) avoids getting stuck on saddle points but can converges slowly (Bottou, 2010) when the minimum lies in a shallow valley, which is likely to occur in over-parameterized models (Dauphin et al., 2014). Momentum, which uses gradients from the previous iterations, can accelerate training (Ruder, 2017) in those cases. Adam (Kingma and Ba, 2017), one of the most widely used momentum-based optimizers, computes an adaptive learning rate for each parameter using the first and second moments from previously computer gradients. This is particularly useful in scenarios where different parameters have different scales, leading to gradients components to have heterogeneous magnitudes. However, despite its popularity and good practical performance, Adam sometimes suffers from poor generalization performance when compared to simpler methods, such as stochastic gradient descent (SGD) with momentum (Wilson et al., 2018). Additionally, its convergence is not guaranteed in certain settings, even in convex optimization problems where it can fail to converge to the optimal solution (Reddi et al., 2019).

Recent advances in optimization theory have shown that second-order methods can improve convergence rates by incorporating curvature information (Vo, 2024; Korbit et al., 2024). However, their prohibitive computational cost has limited practical use in large-scale neural networks. To address this, several approximation strategies have been developed, such as K-FAC (Martens and Grosse, 2020), which uses Kronecker factorization to approximate the Fisher information matrix, and adaptive trust region methods (Vo, 2024) that adjust the region size based on local curvature.

In this paper, we introduce Regression Descent (RD), a novel optimization framework that provides both theoretical insights and practical benefits by reformulating neural network training through the lens of iterative regression. Our approach is motivated by three key observations:

1. In overparameterized networks, the effective dimension of parameter updates is constrained by the data, suggesting that optimization occurs in a lower-dimensional subspace.
2. The linearization inherent in gradient descent can be explicitly leveraged to formulate each step as a well-understood regression problem.
3. Statistical regression techniques offer powerful tools for regularization, conditioning, and theoretical analysis that are underutilized in current optimization methods.

The main contribution of this work is as follows:

1. We propose Regression Descent (RD), an optimization method that reformulates each update step as a regularized least-squares problem in the row space of the Jacobian. This provides a structured and interpretable approach to parameter updates in overparameterized networks.
2. We establish theoretical guarantees for RD, including global convergence to stationary points at a rate of $O(1/k)$ for smooth non-convex objectives. We also show local linear convergence near strict local minima, with explicit dependence on problem conditioning.
3. We design practical variants of RD, including adaptive regularization that adjusts step sizes based on local geometry, as well as mini-batch version that scale to large networks. The method also extends naturally to multi-output and classification settings.
4. We validate RD through experiments, showing that it converges 20–40% faster than SGD and Adam on standard benchmarks. It performs particularly well in difficult regimes such as those involving activation saturation or vanishing gradients.

2 RELATED WORK

The availability of highly efficient computational resources have enabled training increasingly larger models, often containing more parameters than data points. Surprisingly, in such overparameterized settings, gradient descent can still converge to global minima despite the non-convexity of the loss landscape (Du et al., 2019; Allen-Zhu et al., 2019). This has motivated extensive research into understanding the optimization and generalization behavior of these models. In particular, large neural networks often generalize well even when they perfectly fit the training data, challenging traditional statistical wisdom (Belkin et al., 2019; Belkin, 2021; Xu and Gu, 2023). Despite these advances, the detailed training dynamics of deep neural networks remain only partially understood. As neural networks scale to millions or billions of parameters, important questions arise about learning dynamics and the relative importance of different parameters during training. Several studies suggest that many parameters contribute redundantly or have limited impact on training dynamics and final performance. For instance, Sagun et al. (2017); Gur-Ari et al. (2018) showed that optimization largely occurs in a low-dimensional subspace spanned by the leading eigenvectors of the Hessian. Yaras et al. (2023) found that learning primarily occurs within a small invariant subspace of each weight matrix. Likewise,

Kwon et al. (2024) leverage similar insights to compress overparameterized neural networks, while Oymak et al. (2019) employ them to study generalization. Frankle and Carbin (2019) proposed the "Lottery Ticket Hypothesis", which states that a pruned subnetwork can, when initialized correctly, achieve similar accuracy to the original model. Building on this research, the goal is to acknowledge that not all parameters are equally important during training. To formalize this phenomenon, we turn to compressed sensing (Donoho, 2006), which enables recovery of sparse high-dimensional signals from far fewer observations than traditionally required. Our goal is to recover effective parameter updates from a much larger ambient space. This perspective of *low-dimensional learning* motivates an optimization algorithm that exploits the low-dimensional subspace where meaningful learning occurs and provides greater control during training compared to standard gradient descent. Specifically, in a problem with (n) observations and (p) parameters where $(n << p)$, instead of performing a gradient update, we solve an over-parameterized regression problem at each step while applying dimensionality reduction to transform the problem from p-dimensional space to n-dimensional subspace. This approach enables the use of statistical techniques to better understand training dynamics and allows for training in fewer steps.

A similar effort has already been made in Cai et al. (2019), where kernel regression is used at each step instead of gradient descent. The motivation in Cai et al. (2019) to solve a kernel regression problem at every step originates from the idea of the Neural Tangent Kernel (NTK) (Jacot et al., 2020), which states that for sufficiently wide neural networks, the NTK is deterministic (nearly constant) and governs the evolution of neural network training under gradient descent.

However, the motivation of our work differs. In the over-parameterized setting, where the number of parameters p exceeds the number of observations n, the goal is to solve an underdetermined system with more variables than equations. According to the Implicit Function Theorem, the $p - n$ *free parameters* can be locally expressed in terms of the remaining n, reducing the problem to a regression similar to that in Cai et al. (2019). Our approach not only accelerates neural network training but also has the potential to leverage traditional statistical techniques to better understand training dynamics. We further extend the method to handle multivariate output problems.

3 REGRESSION DESCENT

3.1 Preliminary

Definition 1 (Neural Network Regression Problem)**.** Let $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$ with $x^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \mathbb{R}$. We consider neural networks $g : \mathbb{R}^d \times \mathbb{R}^p \to \mathbb{R}$, where $g(x, \theta)$ is the network output for input x and parameters $\theta \in \mathbb{R}^p$. Let $X = ((x^{(1)})^\top, (x^{(2)})^\top, \ldots, (x^{(n)})^\top) \in \mathbb{R}^{n \times d}$ be the matrix of inputs, and $Y = (y^{(1)}, y^{(2)}, \ldots, y^{(n)})^\top \in \mathbb{R}^n$ be the vector of responses, and $g(X, \theta) = (g(x^{(1)}; \theta), \ldots, g(x^{(n)}; \theta))^\top \in \mathbb{R}^n$ be the vector of model predictions. The empirical risk is given by

$$L_n(\theta) = \frac{1}{2n} \sum_{i=1}^{n} (y^{(i)} - g(x^{(i)}, \theta))^2 = \frac{1}{2n} \|Y - g(X, \theta)\|^2$$

Throughout this paper, $\|\cdot\|$ denotes the Euclidean (L_2) norm unless stated otherwise.

Modern machine learning has demonstrated empirically the benefits of using classes of functions $\mathcal{G} = \{g(X; \theta), \theta \in \Theta\}$ with large number of parameters, often taking $p > n$. This may appear problematic to statisticians who focus on elucidating the statistical properties of estimated parameter

$$\hat{\theta} \in \arg\min_{\theta \in \Theta} \frac{1}{2n} \|Y - g(X, \theta)\|^2 \tag{1}$$

and how it relate to the target

$$\theta^* \in \arg\min_{\theta \in \Theta} \frac{1}{2} \mathbb{E}[Y - g(X, \theta)]^2 \tag{2}$$

To analyze the behavior of these over-parameterized models, we consider the local linear approximation of the function class. Denote by

$$D(\theta) = \frac{\partial g(X; \theta)}{\partial \theta}, \tag{3}$$

$$[D(\theta)]_{i,j} = \frac{\partial g(x^{(i)}; \theta)}{\partial \theta_j}, \tag{4}$$

the $n \times p$ matrix of partial derivatives of the model predictions with respect to the parameters.

Using a first-order Taylor series approximation of the model around θ_0, we have

$$g(X, \theta) = g(X, \theta_0) + D(\theta_0)\,(\theta - \theta_0) + o(\|\theta - \theta_0\|) \tag{5}$$

The study of the estimated parameters θ is complicated by the parameters not always being identifiable. Specially, if the rank of $D(\theta^*)$ is $q < p$, which will always occur when $q \leq n < p$, then by implicit function

theorem, we can locally express $p-q$ parameters in as a function of q ones while satisfying $g(X,\theta) = g(X,\theta^*)$. The resolution of this identifiability problem is to focus on understanding the fitted values $g(X,\hat{\theta})$ and how they relate to $g(X,\theta^*)$ instead of analyzing the behaviour of the estimated parameter. To do so, at k^{th} step, we linearize the function around θ_k, which allow us to represent the problem as a regression (least square) problem, and opens opportunities to apply statistical techniques for a deeper understanding of training dynamics. Specifically, we specialize equation 5 to approximate the fitted value of $g(X,\theta)$ in the neighbourhood of θ_k. For notational simplicity, we write $g(\theta) := g(X,\theta)$ and $D_k := D(\theta_k)$ from now on. Then, the first-order Taylor expansion around θ_k becomes:

$$g(\theta) = g(\theta_k) + D_k \cdot (\theta - \theta_k) + o(\|\theta - \theta_k\|) \quad (6)$$

which when we substitute into Equation 1, suggest estimating adjustments $\beta_k = \theta - \theta_k$ to the current parameter θ_k by minimizing

$$\tilde{L}_n(\beta_k) = \frac{1}{2n}\|Y - g(\theta_k) - D_k\beta_k\|^2 \quad (7)$$

Denoting $(Y - g(\theta_k)) = R_k$, which is the residuals when predicting Y by $g(\theta_k)$, we get

$$\tilde{L}_n(\beta_k) = \frac{1}{2n}\|R_k - D_k\beta_k\|^2 \quad (8)$$

We will only consider the situation when $n \leq p$ (Even when this does not hold, it can be enforced by using a minibatch of the data (see Section 5), a practice that is standard in modern ML training). In this case the minimizer of Equation 8 is not unique as the regression problem is underdetermined. For any $\beta_k = \beta_k^r + \beta_k^{r\perp}$ with $\beta_k^r \in rowspan(D_k)$ and $\beta_k^{r\perp} \in rowspan(D_k)^\perp$, we have $D_k\beta_k = D_k\beta_k^r$. Since the loss depends on θ only through $D_k\beta_k$ in the linearized approximation (Eq. (8)), minimizing over the full parameter space is equivalent to minimizing over rowspan(D_k). We leverage this insight to reduce the dimensionality of our regression problem by shifting from estimating p parameters to n, yielding a unique solution. To formalize this, we parameterize $\beta_k = D_k^\top\gamma_k$, which gives

$$\tilde{L}_n(\gamma_k) = \frac{1}{2n}\|R_k - D_kD_k^\top\gamma_k\|^2 \quad (9)$$

Denote $K_k = D_kD_k^\top$, which is a $n \times n$ symmetric matrix. The linearization from Equation 6 is expected to hold for θ near θ_k, or equivalently, for small β_k. We therefore add a penalty $\|\beta_k\|^2 = \|D_k^\top\gamma_k\|^2$ to the loss in Equation 9 to keep the step small. The addition of the penalty term has the added benefit to ensure that there is a unique minimizer of

$$\frac{1}{2n}\|R_k - K_k\gamma_k\|^2 + \frac{c_k}{2}\|D_k^\top\gamma_k\|^2 \quad (10)$$

Algorithm 1: Regression Descent (RD)

Input: Initial θ_0, regularization c_k, data (X, Y)
Output: Trained parameters θ_K
for $k = 0, 1, \ldots, K-1$ **do**
  $R_k \leftarrow Y - g(X,\theta_k)$; `// residuals`
  $D_k \leftarrow \nabla_\theta g(X,\theta_k)$; `// Jacobian`
  $K_k \leftarrow D_kD_k^\top$; `// Gram matrix`
  $\hat{\gamma}_k \leftarrow (K_k + nc_kI)^{-1}R_k$;
  $\theta_{k+1} \leftarrow \theta_k + D_k^\top\hat{\gamma}_k$;
return θ_K

where c_k is the regularization parameter at k^{th} step. The minimizer of equation 10 is

$$\hat{\gamma}_k = (K_k + nc_kI)^{-1}R_k \quad (11)$$

which gives the parameter update rule:

$$\theta_{k+1} = \theta_k + \beta_k \quad (12)$$
$$= \theta_k + D_k^\top(K_k + nc_kI)^{-1}R_k \quad (13)$$

For comparison, the update rule of gradient descent is given by:

$$\theta_{k+1}^{GD} = \theta_k + \alpha_k\frac{1}{n}D(\theta_k)^\top R_k, \quad (14)$$

where α_k is the step size.

The key distinction between equation 13 and equation 14 lies in how they utilize the problem's geometry. Regression descent explicitly accounts for the local geometry of the minimization problem by operating in the effective dimensions. However, this advantage comes with a computational overhead, as it requires inverting an $n \times n$ matrix at each step. Algorithm 1 summarizes Regression Descent. The regularization c_k may be constant or adaptive (Section 6). An extended algorithm covering mini-batch, multi-output, and adaptive regularization variants is provided in the supplement. A natural question is how this geometric perspective extends when the model produces multiple outputs – as in multi-class classification or multi-output regression. We address this next.

4 REGRESSION DESCENT FOR MULTI-CLASS AND MULTI-OUTPUT PROBLEMS

Consider a multi-class classification problem with C classes (the same analysis applies to multi-output regression). For each observation $i = 1, \ldots, n$, let $x^{(i)} \in \mathbb{R}^d$, $y^{(i)} \in \mathbb{R}^C$, and $g(x^{(i)},\theta) \in \mathbb{R}^C$, where $\theta \in \mathbb{R}^p$ denotes the model parameters. The response vector $y^{(i)}$ is one-hot encoded:

$$y^{(ij)} = \begin{cases} 1, & \text{if observation } i \text{ belongs to class } j, \\ 0, & \text{otherwise} \end{cases}$$
$$j = 1, \ldots, C.$$

We collect observations and model outputs into matrices

$$Y = [y^{(1)}, \ldots, y^{(n)}]^\top \in \mathbb{R}^{n \times C},$$
$$g(X, \theta) = [g(x^{(1)}, \theta), \ldots, g(x^{(n)}, \theta)]^\top \in \mathbb{R}^{n \times C}.$$

For each class $j = 1, \ldots, C$, the j-th output is interpreted as the conditional class probability

$$g_j(x^{(i)}, \theta) = p(y^{(i)} = j \mid x^{(i)}).$$

4.1 Empirical Risk and Gradient

The empirical risk on the data $\{(x^{(i)}, y^{(i)})\}_{i=1}^n$ is given by

$$L(\theta) = \frac{1}{2nC} \sum_{i=1}^{n} \sum_{j=1}^{C} \left(y^{(ij)} - g_j(x^{(i)}, \theta) \right)^2. \tag{15}$$

In matrix form, the gradient of the loss function can be written as:

$$\nabla_\theta L(\theta) = -\frac{1}{nC} \sum_{j=1}^{C} D_j(X, \theta)^\top R^{(\cdot j)} \tag{16}$$

where $R^{(\cdot j)} = (R^{(1j)}, \ldots, R^{(nj)})^\top \in \mathbb{R}^n$, and

$$D_j(X, \theta) = \frac{\partial}{\partial \theta} g_j(X, \theta) = (\nabla_\theta g_j(x^{(1)}, \theta), \ldots, \nabla_\theta g_j(x^{(n)}, \theta))^\top \in \mathbb{R}^{n \times p}. \tag{17}$$

Here, $D_j(X, \theta)$ is the Jacobian matrix for the j^{th} output, and $R^{(ij)} = y^{(ij)} - g_j(x^{(i)}, \theta)$ denotes the residual for the i^{th} data point and the j^{th} class.

4.2 Gradient at Iteration k

At the k-th iteration, the gradient is:

$$\nabla_\theta L(\theta_k) = -\frac{1}{nC} \sum_{j=1}^{C} D_j(X, \theta_k)^\top R_k^{(\cdot j)}, \tag{18}$$

where the subscript in $R_k^{(\cdot j)}$ denotes the iteration k. In particular,

$$R_k^{(ij)} = y^{(ij)} - g_j(x^{(i)}, \theta_k). \tag{19}$$

4.3 Update Rule

At iteration k, we linearize the network outputs around the current parameters θ_k and decompose the loss into per-class residual terms (see Appendix 11 for the full derivation). This yields the working objective:

$$\tilde{L}(\gamma_k) = \frac{1}{2nC} \sum_{j=1}^{C} \| R_k^{(\cdot j)} - K_k^{(j)} \gamma_k^{(j)} \|^2$$

where $R_k^{(\cdot j)} = y^{(j)} - g_j(X, \theta_k)$ is the residual for class j, $K_k^{(j)} = D_j(X, \theta_k) \cdot D_j(X, \theta_k)^\top$ is the empirical kernel matrix, and the update is constrained to the row-span of the Jacobian, i.e. $\beta_k^{(j)} = D_j(X, \theta_k)^\top \gamma_k^{(j)}$ with $\gamma_k^{(j)} \in \mathbb{R}^n$. Applying the same reasoning as in Section 3, we obtain the per-class update and its aggregate:

$$\beta_k^{(j)} = D_j(X, \theta_k)^\top \left(K_k^{(j)} + nCc_k I_n \right)^{-1} R_k^{(\cdot j)} \tag{20}$$

$$\beta_k = \frac{1}{C} \sum_{j=1}^{C} \beta_k^{(j)} \tag{21}$$

which gives the parameter update:

$$\theta_{k+1} = \theta_k + \frac{1}{C} \sum_{j=1}^{C} D_j(X, \theta_k)^\top \cdot \left(K_k^{(j)} + nCc_k I_n \right)^{-1} R_k^{(\cdot j)} \tag{22}$$

This update aggregates per-class kernel regression steps, each correcting the residual $R_k^{(\cdot j)}$ in the direction suggested by the j-th class Jacobian. Having established the full-batch update rule, we next consider how this scheme extends to the practical mini-batch setting.

5 MINI-BATCH

In practice, optimization algorithms seldom compute the full gradient, specially in the case of very big network with a large amount of data. Instead, they update the parameters based on a mini-batch, a sample S_k of size $m << n$ from the original data. Denote by

$$D_{S_k} = \left(\frac{\partial g(x^{(i)}; \theta)}{\partial \theta_j} \right)_{x_i \in S_k, j = 1, 2, \ldots, p} \tag{23}$$

the $m \times p$ matrix of partial derivatives for observations in the mini-batch. Following the development

from the proceeding section, we consider minimizing the linearized squared error from Equation 7, but with $\beta_k \in rowspan(D_{S_k})$. This leads us to consider the penalized minimization problem

$$\min_{\gamma \in \mathbb{R}^n} \frac{1}{2}\|R_{S_k} - D_{S_k} D_{S_k}^\top \gamma_k\|^2 + \frac{c_k}{2}\|D_{S_k}^\top \gamma_k\|^2 \quad (24)$$

Denoting $D_{S_k} D_{S_k}^T = Z_{S_k}$, and solving the above minimization problem leads to the following update rule based on minibatch S_k:

$$\theta_{k+1} = \theta_k + D_{S_k}^\top (Z_{S_k} + mc_k I)^{-1} R_{S_k} \quad (25)$$

This requires the calculation of the inverse of a $m \times m$ matrix. The quality of this update, however, depends critically on the choice of c_k, whose careful selection we address next.

6 ADAPTIVE REGULARIZATION PARAMETER

In Regression Descent, c_k at k^{th} step acts as a learning rate analogue and plays a crucial role: a small value of c_k can result in a large update step, potentially moving θ far from its previous value and causing the first-order Taylor approximation to break down. On the other hand, a large c_k may lead to overly conservative updates, offering little to no advantage over traditional gradient descent. In this section, we propose a framework to adaptively select the regularization parameter to enhance convergence and robustness. To begin, we compute certain quantities to assess the quality of the update at step k. However, since the batch X_{S_k} has size $m \ll p$, using it both to take a step and evaluate its effectiveness can lead to overfitting. Instead, we assess the quality of the update using a different batch, say $X_{S_{k+1}}$. Let $R_{S_k} = Y_{S_k} - g(X_{S_k}, \theta_k)$ be the residuals for batch X_{S_k}, and let $R_{S_{k+1}}^{\text{trial}} = Y_{S_{k+1}} - g(X_{S_{k+1}}, \theta_{k+1})$ be the trial residuals for the next batch $X_{S_{k+1}}$, which will also be used in the subsequent step. Then we calculate the *Residual Reduction Ratio*

$$\rho_k = \frac{\|R_{S_{k+1}}^{trial}\|^2}{\|R_{S_k}\|^2} \quad (26)$$

, and *Interpolation parameter*

$$\alpha^* = \frac{-R_{S_k}^\top (R_{S_{k+1}}^{trial} - R_{S_k})}{\|R_{S_{k+1}}^{trial} - R_{S_k}\|^2} \quad (27)$$

Based on these metrics, we adjust c_k according to:

$$c_{k+1} = \begin{cases} c_k/\tau & \text{if } \rho_k < 0.8 \text{ and } \alpha^* > 0.9 \quad \text{(good model)} \\ c_k \nu & \text{if } \rho_k > 0.9 \text{ or } \alpha^* < 0.5 \quad \text{(poor model)} \\ c_k & \text{otherwise} \end{cases}$$

where typically $\nu = 1.8$ and $\tau = 2.7$, following empirically validated values.

To prevent c_k values from exploding, we clamp them between c_{lower} (typically 10^{-7}) and c_{upper} (for classification problems usually set to 1, and for regression problems either 10 or a function of the epoch/iteration count, e.g., $0.1 \times$ epoch).

Empirically, we observed the following:
1. For classification problems with convolutional architectures, a constant value of c_k (usually 0.001) works well.
2. For simple feed-forward architectures, classification problems can also be solved using a constant c_k, which may be scheduled to increase with the epoch, analogous to how learning rate schedulers decrease the step size in Adam or SGD.
3. For regression problems, we strongly recommend using an adaptive regularization technique, since residuals can sometimes explode. In such cases, this strategy prevents unstable updates.

7 CONVERGENCE ANALYSIS

We establish a per-step descent guarantee for RD, showing that the empirical loss decreases at each iteration at a rate governed by the spectrum of K_k and the regularization parameter c_k. We work under the following conditions.

Assumption 1 (Smoothness & Regularity Conditions)**.** The function $g(\cdot, \theta)$ satisfies:

1. The Jacobian $\nabla_\theta g(x, \theta)$ is L_g-Lipschitz continuous in θ.
2. (Spectral bounds) The Gram matrix $K_k = D_k D_k^\top$ satisfies $0 < \lambda_{\min}^* \le \lambda_{\min}^+(K_k) \le \lambda_{\max}(K_k) \le \lambda_{\max}^* < \infty$ for all iterations k, where $\lambda_{\min}^+(K_k) := \min\{\lambda_i(K_k) : \lambda_i(K_k) > 0\}$ denotes the smallest positive eigenvalue.
3. There exists a constant $0 < \alpha \le 1$ such that $\|P_k R_k\| \ge \alpha \|R_k\|$, for all k.
4. The regularization parameters satisfy $c_k \in [c_{\min}, c_{\max}]$ with $0 < c_{\min} < c_{\max} < \infty$.

The spectral bounds (Assumption 1.2) hold when the network maintains sufficient expressivity, as occurs with smooth activations and bounded weights under standard initialization. When K_k becomes near-singular, adaptive regularization increases c_k to maintain stability.

Theorem 1 (Descent Property)**.** Let $L_n(\theta) = \frac{1}{2n}\|Y - g(X, \theta)\|^2$ be the empirical loss, where g has L_g-Lipschitz Jacobian with respect to θ. Define

$$D_k := \nabla_\theta g(X, \theta_k), K_k := D_k D_k^\top, R_k := Y - g(X, \theta_k),$$

and let P_k denote the orthogonal projector onto Range(K_k). Consider the update rule

$$\theta_{k+1} = \theta_k + \beta_k, \quad \text{where } \beta_k = D_k^\top (K_k + nc_k I)^{-1} R_k.$$

Assume that the regularization parameter c_k and the residual satisfy

$$c_k \;\geq\; 2L_g \cdot \kappa(K_k) \cdot \frac{\|R_k\|}{\alpha^2 \sqrt{n}}, \quad \text{where } \kappa(K_k) = \frac{\lambda_{\max}(K_k)}{\lambda^+_{\min}(K_k)}$$

and $\|R_k\| = \sqrt{n}\,\epsilon_k, \quad \epsilon_k = O(1),$ uniformly bounded.

for all k and that the regularization is sufficiently large relative to the spectrum of K_k, i.e., $nc_k \gg \lambda_{\max}(K_k)$ for all k. Then, under Assumption 1, the empirical loss satisfies the descent property:

$$L_n(\theta_{k+1}) - L_n(\theta_k) \;\leq\; -\frac{\lambda^+_{\min}(K_k)}{4\alpha^2 n^2 c_k} \|R_k\|^2 \qquad (28)$$

Proof is provided in APPENDIX.

Theorem 2 (Global Convergence Rate)**.** Assume the loss function $L_n(\theta)$ is bounded below by L_n^* and the conditions of Theorem 1 hold at each iteration, along with Assumptions 1. Then the Regression Descent algorithm satisfies

$$\min_{0 \leq k \leq K-1} \|\nabla L_n(\theta_k)\|^2 \;\leq\; \frac{4\alpha^2 c_{max} \lambda^*_{\max}\big(L_n(\theta_0) - L_n^*\big)}{\lambda^*_{\min} \cdot K}, \qquad (29)$$

establishing an $O(1/K)$ convergence rate to a stationary point.

Proof is provided in APPENDIX.

8 NUMERICAL EXPERIMENTS

In this section, we present numerical experiments where we train neural networks with Regression Descent on both regression and classification tasks. For the regression task, we use data generated from the Lorenz '96 dynamical system (Lorenz, 2006), consisting of 10-dimensional input features X and corresponding outputs y. For the classification tasks, we evaluate on the MNIST (LeCun et al., 1998) and Fashion-MNIST (FMNIST) (Xiao et al., 2017) datasets. MNIST contains 60,000 training examples of handwritten digits (28×28 grayscale images), while FMNIST provides 60,000 grayscale images of fashion articles of the same size.

8.1 Stopping Criteria and Learning Rate

While Mean Squared Error (MSE) is the standard choice in regression, it is less common in classification tasks. However, recent work (Hui and Belkin, 2021) has shown that training neural networks with MSE loss can perform competitively with cross-entropy across a variety of classification benchmarks, suggesting broader applicability for methods such as Regression Descent (RD). That said, driving the MSE all the way to zero is not equally feasible across problems.

In our experiments, we compared RD with SGD, Adam, and KFAC (using the PyTorch implementation of KFAC (Pauloski et al., 2020, 2021)). Since RD involves the inversion of one or more matrices, we chose not to compare the methods in terms of the number of iterations. Instead, for all experiments, we evaluated training error against wall-clock time. With runtime as our primary focus, we adopted a stopping rule based on reaching a small training error that was achievable by all methods, considering the compute budget, as SGD sometimes stagnates. This threshold was determined through a few short exploratory runs. For classification tasks, we additionally required that training accuracy exceed a high value before stopping.

For learning rate selection in SGD, Adam, and KFAC, we used small trial runs to identify a suitable initial value. In regression problems, we also employed a scheduler that gradually decreases the learning rate. For RD, the adaptive regularization strategy described in Section 6 was used in regression problems. In classification settings, especially with convolutional architectures, we found that a constant value of c_k was sufficient.

8.2 Results

For the regression tasks, we employed two fully connected feedforward neural networks with ReLU activations: a deep network with 6 hidden layers and a wide network with a single large hidden layer. A batch size of 100 was used for this task, and figure 1 and 2 illustrate the comparative results.

For MNIST, we employed both a feedforward neural network and a convolutional neural network (CNN), with architectural details provided in Figure 3. In all classification tasks, the networks were trained with a batch size of 256. We observe that RD converges quickly on MNIST for both architectures. All methods were evaluated on held-out test sets, with results and additional experiments provided in the supplementary materials.

8.3 Gradient Saturation

Neural networks often suffer from the vanishing gradient problem, particularly when using activation functions such as sigmoid or tanh (Glorot and Bengio, 2010). These functions saturate at large input mag-

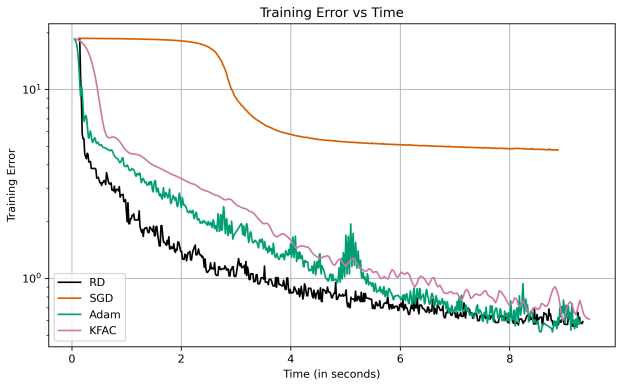


Figure 1: SimpleDeepNN: Fully Connected (FC) layers 10–32–32–32–32–16–8–1 with ReLU, no biases.

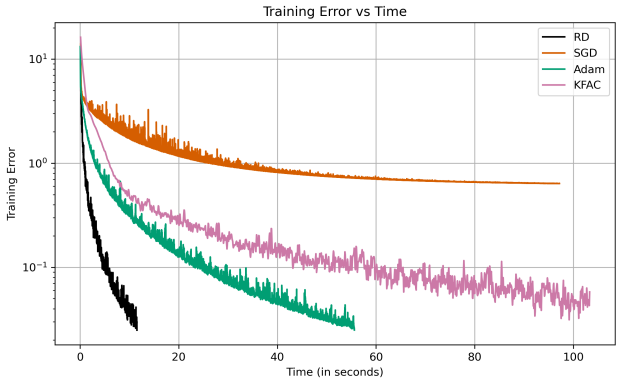


Figure 2: SimpleWideNN: FC layers 10–1032–32–1 with ReLU, no biases.

nitudes, producing near-zero gradients that slow or halt learning in deeper layers. Here, we investigate neural networks using sigmoid activations instead of ReLU to highlight gradient saturation issues (Glorot and Bengio, 2010). Our goal is to observe how Regression Descent performs under conditions where vanishing gradients are more likely. We report results on the Lorenz96 regression task and on MNIST and FMNIST classification tasks, using a simple fully connected feedforward neural network and a small CNN architecture with sigmoid activations, respectively. The results are presented in Figure 4. These preliminary findings suggest that Regression Descent can effectively handle bounded activations; however, we leave a thorough evaluation on additional datasets and architectures for future work.

8.4 Computational Complexity

Proposition 1 (Per-Iteration Complexity)**.** The computational complexity of Regression Descent per iteration is provided in Table 1. where n denotes the full dataset size and m is the mini-batch size.

Operation	Full Batch	Mini-batch
Jacobian computation	$O(np)$	$O(mp)$
Matrix product $K_k = DD^T$	$O(n^2p)$	$O(m^2p)$
Linear system solve	$O(n^3)$	$O(m^3)$
Parameter update	$O(np)$	$O(mp)$
Total	$O(n^2p + n^3)$	$O(m^2p + m^3)$

Table 1: Complexity comparison of full batch vs. mini-batch operations.

Although the $O(m^3)$ cost associated with the linear system solve may appear prohibitive, in typical deep learning settings with $m \in [32, 512]$, this cost remains manageable on modern GPUs. Several factors further support the practical viability of RD:
1. Modern GPUs are highly optimized for dense linear algebra operations, with tensor cores.
2. Improved convergence often compensates for the per-iteration cost.

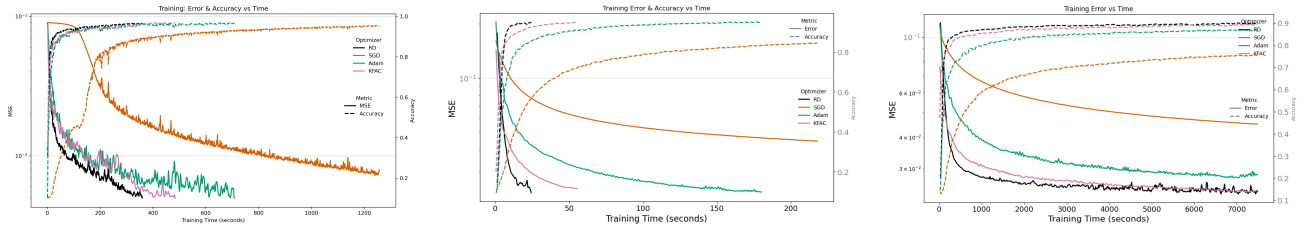
9 DISCUSSION AND FUTURE WORK

We have presented Regression Descent, a novel optimization framework that reconceptualizes neural network training through iterative regression. Our work connects to several recent advances in optimization theory. The low-dimensional structure we exploit relates to recent findings on the effective rank of neural network gradients (Oymak et al., 2019), and our adaptive regularization scheme shares similarities with trust region methods, though it operates in the reduced-dimensional space. These connections suggest that RD naturally adapts to the geometry of neural networks during training.

We acknowledge that applying this method to very large architectures may be challenging, as it requires storing the full Jacobian, which can be infeasible. However, the insights into how network geometry evolves during training point toward natural directions for future research: leveraging this understanding to design more scalable and practical optimization algorithms. While this work focuses on optimization, understanding how Regression Descent affects generalization is a natural and important next step, particularly given the structured, low-dimensional updates it imposes on the network.

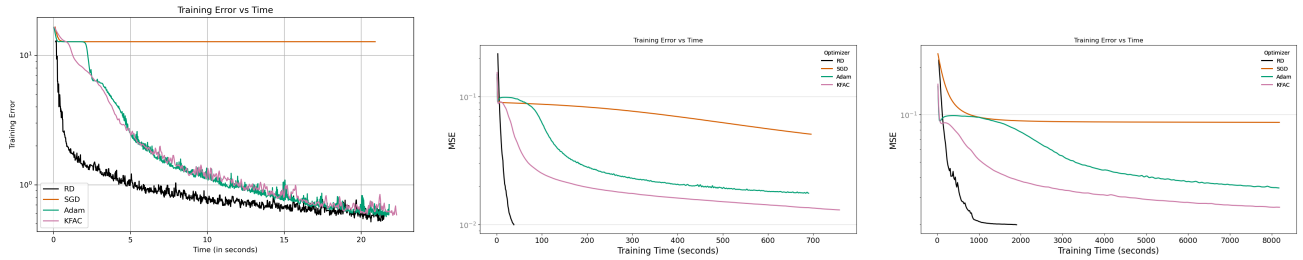
Acknowledgements

This research was funded by Los Alamos National Laboratory's Applied Machine Learning Fellowship and the Laboratory Directed Research and Development (LDRD) Program at Los Alamos National Laboratory under project numbers 20250048DR ("Assured Artifi-



(a) (b) (c)

Figure 3: (a) *Fully connected model on MNIST* — Flatten(1×28×28 → 784) → FC(784 → 128 → 64 → 32 → 10), with ReLU activations and softmax output; (b) *SmallCNN on MNIST* — Input(1×28×28) → Conv2D(1, 8, 3) → ReLU → MaxPool2D(2×2) → Flatten(8×13×13) → FC(10); (c) *TinyCNN on FMNIST* — Input(1×28×28) → Conv2D(1, 16, 3, padding=1) → ReLU → MaxPool2D(2×2) → Flatten(16×14×14) → FC(10).



(a) (b) (c)

Figure 4: (a) *SimpleDeepNN on Lorenz96: FC layers 10–32–32–32–32–16–8–1 with* ***Sigmoid*** *activations;* (b) *SmallCNN on MNIST* — Input(1×28×28) → Conv2D(1, 8, 3) → **Sigmoid** → MaxPool2D(2×2) → Flatten(8×13×13) → FC(10) → **Sigmoid**; (c) *SmallCNN on FMNIST* (same architecture as (b))

cial Intelligence: Efficient and Scalable Guardrail Certification for Science and Security") and 20240734DI ("Artificial Intelligence for Mission (ArtIMis)"). Los Alamos National Laboratory is operated by Triad National Security, LLC, for the National Nuclear Security Administration of the U.S. Department of Energy (Contract No. 89233218CNA000001).

References

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 242–252. PMLR, 09–15 Jun 2019. URL `https://proceedings.mlr.press/v97/allen-zhu19a.html`.

Mikhail Belkin. Fit without fear: remarkable mathematical phenomena of deep learning through the prism of interpolation, 2021. URL `https://arxiv.org/abs/2105.14368`.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116 (32):15849–15854, July 2019. ISSN 1091-6490. doi:10.1073/pnas.1903070116. URL `http://dx.doi.org/10.1073/pnas.1903070116`.

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In Yves Lechevallier and Gilbert Saporta, editors, *Proceedings of COMPSTAT'2010*, pages 177–186, Heidelberg, 2010. Physica-Verlag HD. ISBN 978-3-7908-2604-3.

Tianle Cai, Ruiqi Gao, Jikai Hou, Siyu Chen, Dong Wang, Di He, Zhihua Zhang, and Liwei Wang. Gram-gauss-newton method: Learning overparameterized neural networks for regression problems, 2019. URL `https://arxiv.org/abs/1905.11675`.

Yann Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, 2014. URL `https://arxiv.org/abs/1406.2572`.

D.L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006. doi:10.1109/TIT.2006.871582.

Simon S. Du, Jason D. Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks, 2019. URL `https://arxiv.org/abs/1811.03804`.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks, 2019. URL `https://arxiv.org/abs/1803.03635`.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on*

Artificial Intelligence and Statistics, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL https://proceedings.mlr.press/v9/glorot10a.html.

Guy Gur-Ari, Daniel A. Roberts, and Ethan Dyer. Gradient descent happens in a tiny subspace, 2018. URL https://arxiv.org/abs/1812.04754.

Like Hui and Mikhail Belkin. Evaluation of neural architectures trained with square loss vs cross-entropy in classification tasks, 2021. URL https://arxiv.org/abs/2006.07322.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks, 2020. URL https://arxiv.org/abs/1806.07572.

Nikhil Ketkar. *Stochastic Gradient Descent*, pages 113–132. Apress, Berkeley, CA, 2017. ISBN 978-1-4842-2766-4. doi:10.1007/978-1-4842-2766-4_8. URL https://doi.org/10.1007/978-1-4842-2766-4_8.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Mikalai Korbit, Adeyemi D. Adeoye, Alberto Bemporad, and Mario Zanon. Exact gauss-newton optimization for training deep neural networks, 2024. URL https://arxiv.org/abs/2405.14402.

Soo Min Kwon, Zekai Zhang, Dogyoon Song, Laura Balzano, and Qing Qu. Efficient compression of overparameterized deep models through low-dimensional learning dynamics, 2024. URL https://arxiv.org/abs/2311.05061.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Edward N. Lorenz. *Predictability – a problem partly solved*, page 40–58. Cambridge University Press, 2006.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature, 2020. URL https://arxiv.org/abs/1503.05671.

Samet Oymak, Zalan Fabian, Mingchen Li, and Mahdi Soltanolkotabi. Generalization guarantees for neural networks via harnessing the low-rank structure of the jacobian, 2019. URL https://arxiv.org/abs/1906.05392.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS 2017 Workshop on Autodiff*, 2017. URL https://openreview.net/forum?id=BJJsrmfCZ.

J. Gregory Pauloski, Zhao Zhang, Lei Huang, Weijia Xu, and Ian T. Foster. Convolutional Neural Network Training with Distributed K-FAC. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20. IEEE Press, 2020. ISBN 9781728199986. doi:10.5555/3433701.3433826.

J. Gregory Pauloski, Qi Huang, Lei Huang, Shivaram Venkataraman, Kyle Chard, Ian Foster, and Zhao Zhang. Kaisa: An Adaptive Second-Order Optimizer Framework for Deep Neural Networks. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384421. doi:10.1145/3458817.3476152. URL https://doi.org/10.1145/3458817.3476152.

Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond, 2019. URL https://arxiv.org/abs/1904.09237.

Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017. URL https://arxiv.org/abs/1609.04747.

Levent Sagun, Leon Bottou, and Yann LeCun. Eigenvalues of the hessian in deep learning: Singularity and beyond, 2017. URL https://arxiv.org/abs/1611.07476.

James Vo. Efficient second-order neural network optimization via adaptive trust region methods, 2024. URL https://arxiv.org/abs/2410.02293.

Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning, 2018. URL https://arxiv.org/abs/1705.08292.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Xingyu Xu and Yuantao Gu. Benign overfitting of non-smooth neural networks beyond lazy training. In Francisco Ruiz, Jennifer Dy, and Jan-Willem van de Meent, editors, *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, volume 206 of *Proceedings of Machine Learning Research*, pages 11094–11117. PMLR, 25–27 Apr 2023. URL https://proceedings.mlr.press/v206/xu23k.html.

Can Yaras, Peng Wang, Wei Hu, Zhihui Zhu, Laura Balzano, and Qing Qu. Invariant low-dimensional subspaces in gradient descent for learning deep matrix factorizations. In *NeurIPS 2023 Workshop on Mathematics of Modern Machine Learning*, 2023. URL https://openreview.net/forum?id=4pPnQqUMLS.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. Yes
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. Yes
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. Yes

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. Yes

(b) Complete proofs of all theoretical results. Yes (Proofs are in Appendix)

(c) Clear explanations of any assumptions. Yes

3. For all figures and tables that present empirical results, check if you include:

(a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). Yes

(b) All the training details (e.g., data splits, hyperparameters, how they were chosen). Yes

(c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). No. (Due to computational constraints, we report results from a single run and do not include error bars across multiple random seeds.)

(d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). No

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:

(a) Citations of the creator If your work uses existing assets. Yes

(b) The license information of the assets, if applicable. Not Applicable

(c) New assets either in the supplemental material or as a URL, if applicable. Not Applicable

(d) Information about consent from data providers/curators. Not Applicable (All the data used is publicly available)

(e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. Not Applicable

5. If you used crowdsourcing or conducted research with human subjects, check if you include:

(a) The full text of instructions given to participants and screenshots. Not Applicable

(b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. Not Applicable

(c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. Not Applicable

Supplementary Materials

10 APPENDIX

Proof of Theorem 1. We first write the Taylor series expansion of the estimator $g(X, \theta)$ at θ_{k+1} around θ_k:

$$g(X, \theta_{k+1}) = g(X, \theta_k) + D_k \beta_k + r_k, \tag{30}$$

where $D_k = \nabla_\theta g(X, \theta_k)$ is the Jacobian and r_k is the remainder term. Using the Lipschitz continuity of the Jacobian with constant L_g, the remainder can be bounded as

$$\|r_k\| \leq \frac{\sqrt{n} L_g}{2} \|\beta_k\|^2 \tag{31}$$

Subtracting Y from both sides of equation (30) gives

$$Y - g(X, \theta_{k+1}) = R_k - D_k \beta_k - r_k \tag{32}$$

The squared norm of the new residual is

$$\begin{aligned} \|Y - g(X, \theta_{k+1})\|^2 &= \|R_k - D_k \beta_k - r_k\|^2 \\ &= \|R_k - D_k \beta_k\|^2 \\ &\quad - 2\langle R_k - D_k \beta_k, r_k \rangle + \|r_k\|^2 \end{aligned}$$

The change in empirical loss is given by

$$\begin{aligned} L_n(\theta_{k+1}) - L_n(\theta_k) &= \frac{1}{2n} \Big(\|R_k - D_k \beta_k\|^2 - \|R_k\|^2 \Big) \\ &\quad - \frac{1}{n} \langle R_k - D_k \beta_k, r_k \rangle + \frac{1}{2n} \|r_k\|^2 \\ &= -\frac{1}{n} \langle R_k, D_k \beta_k \rangle + \frac{1}{2n} \|D_k \beta_k\|^2 \\ &\quad - \frac{1}{n} \langle R_k - D_k \beta_k, r_k \rangle + \frac{1}{2n} \|r_k\|^2 \end{aligned}$$

Using

$$D_k \beta_k = K_k (K_k + n c_k I)^{-1} R_k,$$

we have

$$\langle R_k, D_k \beta_k \rangle = R_k^\top K_k (K_k + n c_k I)^{-1} R_k \tag{33}$$

$$\|D_k \beta_k\|^2 = R_k^\top (K_k + n c_k I)^{-1} K_k^2 (K_k + n c_k I)^{-1} R_k \tag{34}$$

Substituting these gives

$$\begin{aligned} L_n(\theta_{k+1}) - L_n(\theta_k) &= -\frac{1}{n} R_k^\top K_k (K_k + n c_k I)^{-1} R_k \\ &\quad + \frac{1}{2n} R_k^\top (K_k + n c_k I)^{-1} K_k^2 (K_k + n c_k I)^{-1} R_k \\ &\quad - \frac{1}{n} \langle R_k - D_k \beta_k, r_k \rangle + \frac{1}{2n} \|r_k\|^2 \end{aligned} \tag{35}$$

From Lemma 2, we have

$$\begin{aligned} -\frac{1}{n} R_k^\top K_k (K_k + nc_k I)^{-1} R_k \\ +\frac{1}{2n} R_k^\top (K_k + nc_k I)^{-1} K_k^2 (K_k + nc_k I)^{-1} R_k \\ \leq -\frac{1}{2n} \frac{\lambda_{\min}^+(K_k)}{\lambda_{\min}^+(K_k) + nc_k} \|P_k R_k\|^2, \end{aligned} \tag{36}$$

and from Lemma 3,

$$\frac{1}{2n}\|r_k\|^2 \leq \frac{L_g^2}{8} \left(\frac{\lambda_{\max}(K_k)}{(\lambda_{\max}(K_k) + nc_k)^2} \right)^2 \|R_k\|^4 \tag{37}$$

Moreover, using Lemma 4,

$$\begin{aligned} -\frac{1}{n} \langle R_k - D_k \beta_k,\, r_k \rangle \;\leq\; & \frac{c_k L_g \sqrt{n}\, \lambda_{\max}(K_k)}{2(\lambda_{\min}(K_k) + nc_k)} \\ & \times \frac{1}{(\lambda_{\max}(K_k) + nc_k)^2} \|R_k\|^3. \end{aligned} \tag{38}$$

Thus, using Equations (36), (37), and (38), we obtain

$$\begin{aligned} L_n(\theta_{k+1}) - L_n(\theta_k) \leq & -\frac{1}{2n} \frac{\lambda_{\min}^+(K_k)}{\lambda_{\min}^+(K_k) + nc_k} \|P_k R_k\|^2 \\ & + \frac{c_k L_g \sqrt{n} \lambda_{\max}(K_k)}{2(\lambda_{\min}(K_k) + nc_k)(\lambda_{\max}(K_k) + nc_k)^2} \|R_k\|^3 \\ & + \frac{L_g^2}{8} \left(\frac{\lambda_{\max}(K_k)}{(\lambda_{\max}(K_k) + nc_k)^2} \right)^2 \|R_k\|^4 \end{aligned} \tag{39}$$

We assume that R_k is not nearly orthogonal to Range(K_k), i.e., there exists a constant $0 < \alpha \leq 1$ such that $\|P_k R_k\| \geq \alpha \|R_k\|$ throughout the convergence basin. This is natural away from stationary points: when $\nabla L_n(\theta_k) = -\frac{1}{n} D_k^\top R_k \neq 0$, we have $P_k R_k \neq 0$. Near convergence where the gradient becomes small, both $\|P_k R_k\|$ and $\|R_k\|$ approach zero together. Now, we have $\|R_k\| = \sqrt{n}\,\epsilon_k$ with $\epsilon_k = O(1)$ and $nc_k \gg \lambda_{\max}(K_k) \geq \lambda_{\min}^+(K_k)$. Then each term scales as

$$\begin{aligned} \text{Term 1:} \quad & -\frac{\lambda_{\min}^+(K_k)}{2n^2 c_k} \|P_k R_k\|^2 \leq -\frac{\lambda_{\min}^+(K_k)\alpha^2}{2n^2 c_k} \|R_k\|^2 \\ & = -\frac{C_1 \epsilon_k^2}{n}, \\ \text{Term 2:} \quad & \frac{L_g \lambda_{\max}(K_k)}{2n^{5/2} c_k^2} \|R_k\|^3 = \frac{C_2 \epsilon_k^3}{n}, \\ \text{Term 3:} \quad & \frac{L_g^2 \lambda_{\max}^2(K_k)}{8n^4 c_k^4} \|R_k\|^4 = \frac{C_3 \epsilon_k^4}{n^2}, \end{aligned}$$

where

$$\begin{aligned} C_1 &= \frac{\lambda_{\min}^+(K_k)\alpha^2}{2c_k} \\ C_2 &= \frac{L_g \lambda_{\max}(K_k)}{2c_k^2}, \\ C_3 &= \frac{L_g^2 \lambda_{\max}^2(K_k)}{8c_k^4}. \end{aligned}$$

For large n, the dominant terms are $O(n^{-1})$. We require

$$C_1 \epsilon_k^2 > C_2 \epsilon_k^3 + \frac{C_3 \epsilon_k^4}{n}.$$

For small ϵ_k, this holds if

$$\epsilon_k < \frac{C_1}{C_2} = \frac{\lambda^+_{\min}(K_k) c_k \alpha^2}{L_g \lambda_{\max}(K_k)} \tag{40}$$

Thus the convergence basin is

$$\|R_k\| < \frac{\lambda^+_{\min}(K_k) c_k \alpha^2}{L_g \lambda_{\max}(K_k)} \cdot \sqrt{n} \tag{41}$$

Remark 1 (Convergence Basin)**.** The convergence basin characterizes the region where the algorithm is guaranteed to make progress. When $\|R_k\|$ satisfies this bound, the descent terms dominate the higher-order remainder terms, ensuring loss reduction. This provides a precise characterization of the algorithm's region of effectiveness.

For sufficiently large n and small ϵ_k,

$$\begin{aligned} L_n(\theta_{k+1}) - L_n(\theta_k) &\leq -\frac{C_1 \epsilon_k^2}{n} + \frac{C_2 \epsilon_k^3}{n} + \frac{C_3 \epsilon_k^4}{n^2} \\ &\leq -\frac{C_1 \epsilon_k^2}{2n} \\ &< 0 \end{aligned} \tag{42}$$

where the second inequality holds when

$$\epsilon_k < \frac{C_1}{2C_2}. \tag{43}$$

which is always true when

$$c_k \;\geq\; 2L_g \cdot \kappa(K_k) \cdot \frac{\|R_k\|}{\alpha^2 \sqrt{n}} \tag{44}$$

For sufficiently small ϵ_k and large n, we have

$$\begin{aligned} L_n(\theta_{k+1}) - L_n(\theta_k) &\leq -\frac{C_1 \epsilon_k^2}{2n} < 0 \\ &\Longleftrightarrow \\ L_n(\theta_{k+1}) - L_n(\theta_k) &\leq -\frac{\alpha^2 \lambda^+_{\min}(K_k)}{4n^2 c_k} \|R_k\|^2 \end{aligned} \tag{45}$$

This establishes the descent property of the algorithm.

□

Proof of Theorem 2. From theorem 1 (descent property), we have

$$L_n(\theta_{k+1}) \;\leq\; L_n(\theta_k) - \frac{1}{4n^2} \frac{\lambda^+_{\min}(K_k)}{\alpha^2 c_k} \|R_k\|^2 \tag{46}$$

Since $\nabla L_n(\theta) = -\frac{1}{n} D(\theta)^\top R(\theta)$, it follows that

$$\|\nabla L_n(\theta_k)\|^2 = \frac{1}{n^2} \|D_k^\top R_k\|^2 \;\leq\; \frac{\lambda_{\max}(K_k)}{n^2} \|R_k\|^2 \tag{47}$$

Therefore,

$$\|R_k\|^2 \;\geq\; \frac{n^2}{\lambda_{\max}(K_k)} \|\nabla L_n(\theta_k)\|^2 \tag{48}$$

Substituting this into the descent inequality gives

$$L_n(\theta_{k+1}) \;\leq\; L_n(\theta_k) - \frac{\lambda^+_{\min}(K_k)}{4 c_k \alpha^2 \lambda_{\max}(K_k)} \|\nabla L_n(\theta_k)\|^2 \tag{49}$$

Summing over $k = 0, \ldots, K-1$ yields

$$L_n(\theta_K) - L_n(\theta_0) \;\le\; -\sum_{k=0}^{K-1} \frac{\lambda^+_{\min}(K_k)}{4c_k\alpha^2\,\lambda_{\max}(K_k)}\,\|\nabla L_n(\theta_k)\|^2 \tag{50}$$

Since $L_n(\theta_0) \ge L_n^*$ for all k, we can write

$$\sum_{k=0}^{K-1} \frac{\lambda^+_{\min}(K_k)}{4c_k\alpha^2\,\lambda_{\max}(K_k)}\,\|\nabla L_n(\theta_k)\|^2 \;\le\; L_n(\theta_0) - L_n^* \tag{51}$$

Using the bounds $c_k \le c_{\max}$, $\lambda^+_{\min}(K_k) \ge \lambda^*_{\min}$, and $\lambda_{\max}(K_k) \le \lambda^*_{\max}$ for all k, and noting that

$$\min_{0\le k\le K-1} \|\nabla L_n(\theta_k)\|^2 \;\le\; \frac{1}{K}\sum_{k=0}^{K-1} \|\nabla L_n(\theta_k)\|^2 \tag{52}$$

we obtain

$$\min_{0\le k\le K-1} \|\nabla L_n(\theta_k)\|^2 \;\le\; \frac{4c_{max}\alpha^2\lambda^*_{\max}\big(L_n(\theta_0) - L_n^*\big)}{K\lambda^*_{\min}}. \tag{53}$$

□

Lemma 1 (Error Term is Smaller than Descent Term)**.** Let $R_k \in \mathbb{R}^n$ be the residual vector, and let $K_k = D_k D_k^\top$ be a symmetric positive semidefinite matrix. Then the two terms

$$A := \frac{1}{n} R_k^\top K_k (K_k + nc_k I)^{-1} R_k$$

$$B := \frac{1}{2n} R_k^\top (K_k + nc_k I)^{-1} K_k^2 (K_k + nc_k I)^{-1} R_k$$

satisfy the inequality:

$$B \le \frac{1}{2} A$$

Proof. Since K_k is symmetric and PSD, it admits an eigen-decomposition:

$$K_k = U\Lambda U^\top, \quad \text{with } \Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n),\ \lambda_i \ge 0$$

Let $z := U^\top R_k$ be the projection of R_k onto the eigenbasis. Then we have:

$$\begin{aligned} A &= \frac{1}{n} R_k^\top K_k (K_k + nc_k I)^{-1} R_k \\ &= \frac{1}{n} z^\top \Lambda (\Lambda + nc_k I)^{-1} z = \frac{1}{n}\sum_{i=1}^n \frac{\lambda_i}{\lambda_i + nc_k} z_i^2 \\ B &= \frac{1}{2n} R_k^\top (K_k + nc_k I)^{-1} K_k^2 (K_k + nc_k I)^{-1} R_k \\ &= \frac{1}{2n} z^\top \frac{\Lambda^2}{(\Lambda + nc_k I)^2} z = \frac{1}{2n}\sum_{i=1}^n \frac{\lambda_i^2}{(\lambda_i + nc_k)^2} z_i^2 \end{aligned}$$

Now observe that for each i:

$$\frac{\lambda_i^2}{(\lambda_i + nc_k)^2} \le \frac{\lambda_i}{\lambda_i + nc_k}$$

which implies:

$$\frac{1}{2n}\frac{\lambda_i^2}{(\lambda_i + nc_k)^2} z_i^2 \le \frac{1}{2n}\frac{\lambda_i}{\lambda_i + nc_k} z_i^2 = \frac{1}{2}\cdot\left(\frac{1}{n}\frac{\lambda_i}{\lambda_i + nc_k} z_i^2\right)$$

Summing over i gives:

$$B \leq \frac{1}{2}A$$

□

Lemma 2 (Upper Bound on Descent Term)**.** Let $R_k \in \mathbb{R}^n$ be the residual vector, and let $K_k = D_k D_k^\top$ be a symmetric positive semidefinite (PSD) matrix. Let P_k be the orthogonal projector onto Range(K_k) and $\lambda_{\min}^+$ be the smallest positive eigenvalue of K_k. Then the two terms

$$A := \frac{1}{n} R_k^\top K_k (K_k + nc_k I)^{-1} R_k,$$

$$B := \frac{1}{2n} R_k^\top (K_k + nc_k I)^{-1} K_k^2 (K_k + nc_k I)^{-1} R_k$$

satisfy the inequality:

$$B \leq \frac{1}{2}A$$

Furthermore, the term $-A + B$ is bounded from above by:

$$-A + B \leq -\frac{1}{2n} \frac{\lambda_{\min}^+}{\lambda_{\min}^+ + nc_k} \|P_k R_k\|^2$$

Proof. Since K_k is symmetric and PSD, it admits an eigen-decomposition:

$$K_k = U \Lambda U^\top, \quad \text{with } \Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n),\ \lambda_i \geq 0$$

Let $z := U^\top R_k$ be the projection of R_k onto the eigenbasis. Then we have:

$$\begin{aligned} A &= \frac{1}{n} R_k^\top K_k (K_k + nc_k I)^{-1} R_k = \frac{1}{n} \sum_{i=1}^n \frac{\lambda_i}{\lambda_i + nc_k} z_i^2 \\ B &= \frac{1}{2n} R_k^\top (K_k + nc_k I)^{-1} K_k^2 (K_k + nc_k I)^{-1} R_k \\ &= \frac{1}{2n} \sum_{i=1}^n \frac{\lambda_i^2}{(\lambda_i + nc_k)^2} z_i^2 \end{aligned}$$

For each eigenvalue $\lambda_i \geq 0$, the inequality $\frac{\lambda_i^2}{(\lambda_i + nc_k)^2} \leq \frac{\lambda_i}{\lambda_i + nc_k}$ holds. This directly implies that $B \leq \frac{1}{2}A$.

Now, let's find the upper bound for $-A + B$. We start with the inequality derived from the first part of the proof:

$$B - A \leq \frac{1}{2}A - A = -\frac{1}{2}A$$

To find a bound for $-A/2$, we need to find a lower bound for A. Since the function $f(\lambda) = \frac{\lambda}{\lambda + nc_k}$ is monotonically increasing with λ, its minimum value for $\lambda \geq \lambda_{\min}^+$ is $\frac{\lambda_{\min}^+}{\lambda_{\min}^+ + nc_k}$.

Restricting the sum to positive eigenvalues (i.e., the range of K_k):

$$\begin{aligned} A &= \frac{1}{n} \sum_{i=1}^n \frac{\lambda_i}{\lambda_i + nc_k} z_i^2 \\ &\geq \frac{1}{n} \sum_{i:\lambda_i > 0} \left(\frac{\lambda_{\min}^+}{\lambda_{\min}^+ + nc_k} \right) z_i^2 \\ &= \frac{1}{n} \frac{\lambda_{\min}^+}{\lambda_{\min}^+ + nc_k} \sum_{i:\lambda_i > 0} z_i^2 \end{aligned}$$

Since P_k projects onto Range(K_k), we have $\sum_{i:\lambda_i>0} z_i^2 = \|P_k R_k\|^2$, giving:

$$A \geq \frac{1}{n} \frac{\lambda_{\min}^+}{\lambda_{\min}^+ + nc_k} \|P_k R_k\|^2$$

Multiplying this inequality by $-\frac{1}{2}$ reverses the inequality sign, giving an upper bound for $-A/2$:

$$-\frac{1}{2}A \leq -\frac{1}{2n} \frac{\lambda_{\min}^+}{\lambda_{\min}^+ + nc_k} \|P_k R_k\|^2$$

Combining this with our first inequality, $B - A \leq -A/2$, we get the final result:

$$B - A \leq -\frac{1}{2n} \frac{\lambda_{\min}^+}{\lambda_{\min}^+ + nc_k} \|P_k R_k\|^2$$

□

Lemma 3 (Taylor Remainder Bound)**.** Let r_k be the remainder term from the first-order Taylor expansion, and let R_k be the residual vector. Let $c_k > 0$ be a damping parameter and L_g the Lipschitz constant of the Jacobian. If $\lambda_{\max}(K_k)$ denotes the largest eigenvalue of $K_k = D_k D_k^\top$, then

$$\begin{aligned} \frac{1}{2n}\|r_k\|^2 &\leq \frac{L_g^2}{8} \left(\frac{\lambda_{\max}(K_k)}{(\lambda_{\max}(K_k) + nc_k)^2} \right)^2 \|R_k\|^4 \\ &= \frac{L_g^2 \left(\lambda_{\max}(K_k)\right)^2}{8(\lambda_{\max}(K_k) + nc_k)^4} \|R_k\|^4 \end{aligned} \tag{54}$$

Proof. We begin with the standard bound on the squared norm of the Taylor remainder:

$$\|r_k\|^2 \leq \frac{nL_g^2}{4} \|\beta_k\|^4.$$

Hence,

$$\frac{1}{2n}\|r_k\|^2 \leq \frac{1}{2n} \cdot \frac{nL_g^2}{4} \|\beta_k\|^4 = \frac{L_g^2}{8} \|\beta_k\|^4.$$

Next, recall the spectral form of $\|\beta_k\|^2$:

$$\begin{aligned} \|\beta_k\|^2 &= R_k^\top (K_k + nc_k I)^{-1} K_k (K_k + nc_k I)^{-1} R_k \\ &= \sum_{i=1}^{n} \frac{\lambda_i}{(\lambda_i + nc_k)^2} (v_i^\top R_k)^2 \end{aligned} \tag{55}$$

where $\{\lambda_i\}$ are the eigenvalues of K_k with eigenvectors $\{v_i\}$. Thus,

$$\begin{aligned} \|\beta_k\|^2 &\leq \max_i \frac{\lambda_i}{(\lambda_i + nc_k)^2} \|R_k\|^2 \\ &\leq \frac{\lambda_{\max}(K_k)}{(\lambda_{\max}(K_k) + nc_k)^2} \|R_k\|^2. \end{aligned} \tag{56}$$

Squaring both sides gives

$$\|\beta_k\|^4 \leq \left(\frac{\lambda_{\max}(K_k)}{(\lambda_{\max}(K_k) + nc_k)^2} \right)^2 \|R_k\|^4 \tag{57}$$

Substituting into the earlier inequality yields

$$\frac{1}{2n}\|r_k\|^2 \leq \frac{L_g^2}{8} \left(\frac{\lambda_{\max}(K_k)}{(\lambda_{\max}(K_k) + nc_k)^2} \right)^2 \|R_k\|^4,$$

which completes the proof. □

Lemma 4 (Cross-Term Bound via Cauchy-Schwarz)**.** Let R_k be the residual, D_k the Jacobian matrix, $\beta_k = D_k^\top (K_k + nc_k I)^{-1} R_k$ the update step, and r_k the Taylor remainder. Then:

$$\left|\frac{1}{n}\langle R_k - D_k\beta_k, r_k\rangle\right| \;\le\; \frac{c_k L_g \sqrt{n}\,\lambda_{\max}(K_k)}{2(\lambda_{\min}(K_k) + nc_k)} \times \frac{1}{(\lambda_{\max}(K_k) + nc_k)^2}\,\|R_k\|^3.$$

Proof. By Cauchy-Schwarz inequality:

$$\left|\frac{1}{n}\langle R_k - D_k\beta_k, r_k\rangle\right| \le \frac{1}{n}\|R_k - D_k\beta_k\| \cdot \|r_k\|$$

First, we bound $\|R_k - D_k\beta_k\|$. Since $D_k\beta_k = K_k(K_k + nc_k I)^{-1}R_k$, we have:

$$\begin{aligned} R_k - D_k\beta_k &= R_k - K_k(K_k + nc_k I)^{-1}R_k \\ &= \left[I - K_k(K_k + nc_k I)^{-1}\right]R_k \\ &= nc_k(K_k + nc_k I)^{-1}R_k \end{aligned} \tag{58}$$

Therefore:

$$\begin{aligned} \|R_k - D_k\beta_k\| &= nc_k\,\|(K_k + nc_k I)^{-1}R_k\| \\ &\le \frac{nc_k}{\lambda_{\min}(K_k) + nc_k}\,\|R_k\| \end{aligned} \tag{59}$$

Next, using the bound on $\|r_k\|$ and the bound on $\|\beta_k\|^2$ from Lemma 3:

$$\|r_k\| \le \frac{\sqrt{n}L_g}{2}\|\beta_k\|^2 \le \frac{\sqrt{n}L_g}{2} \cdot \frac{\lambda_{\max}(K_k)}{(\lambda_{\max}(K_k) + nc_k)^2}\|R_k\|^2 \tag{60}$$

Combining these bounds:

$$\begin{aligned} \left|\frac{1}{n}\langle R_k - D_k\beta_k, r_k\rangle\right| &\le \frac{1}{n} \cdot \frac{nc_k}{\lambda_{\min}(K_k) + nc_k}\|R_k\| \cdot \frac{\sqrt{n}L_g}{2} \\ &\quad \cdot \frac{\lambda_{\max}(K_k)}{(\lambda_{\max}(K_k) + nc_k)^2}\|R_k\|^2 \\ &= \frac{c_k L_g \sqrt{n}\lambda_{\max}(K_k)}{2(\lambda_{\min}(K_k) + nc_k)(\lambda_{\max}(K_k) + nc_k)^2}\|R_k\|^3 \end{aligned}$$

□

11 Loss Expansion for the Update Rule of Multi-Class Problems

Starting from the per-sample squared loss, we have:

$$\begin{aligned} L(\theta) &= \frac{1}{2nC}\sum_{i=1}^{n}\sum_{j=1}^{C}\left(y^{(ij)} - g_j(x^{(i)};\theta)\right)^2 \\ &= \frac{1}{2nC}\sum_{j=1}^{C}\left\|y^{(j)} - g_j(X,\theta)\right\|^2 \\ &= \frac{1}{2nC}\sum_{j=1}^{C}\left\|R_k^{(\cdot j)} - \left(g_j(X,\theta) - g_j(X,\theta_k)\right)\right\|^2 \\ &= \frac{1}{2nC}\sum_{j=1}^{C}\left\|R_k^{(\cdot j)} - D_j(X,\theta_k)\cdot\beta_k^{(j)}\right\|^2 \end{aligned} \tag{61}$$

where the last step applies a first-order linearization of $g_j(X,\theta)$ around θ_k.

12 EXTENDED ALGORITHM: UNIFIED REGRESSION DESCENT

This section presents a comprehensive algorithmic description of Regression Descent that unifies the full-batch, mini-batch, multi-output, and adaptive regularization variants described in the main paper. Algorithm 2 provides the complete procedure with conditional branches for each variant.

The system $(K_k + n_k c_k I)\hat{\gamma}_k = R_k$ can be solved via Cholesky decomposition in $O(n_k^3)$ time, or iteratively via conjugate gradient for larger n_k.

13 Comparative Analysis of Gradient Descent and Regression Descent

For a regression problem, at the k^{th} step, the update rule for regression descent is given by:

$$\theta_{k+1} = \theta_k + \beta_k \tag{62}$$
$$= \theta_k + D_k^\top (K_k + nc_k I)^{-1} R_k \tag{63}$$

$$g(\theta) = g(\theta_k) + D_k(\theta - \theta_k) + o(\|\theta - \theta_k\|) \tag{64}$$

which after substituting in Equation 64, reveals that the fitted values satisfies a generalized gradient descent with

$$\widehat{g(\theta_{k+1})} = g(\theta_k) + K_k(K_k + nc_k I)^{-1} R_k \tag{65}$$

However, in gradient descent, we have

$$\nabla_\theta L(\theta_k) = -\frac{1}{n} D(\theta_k)^\top (Y - g(\theta_k)) = -\frac{1}{n} D(\theta_k)^\top R_k \tag{66}$$

and, gradient descent updates the parameters using the recursive rule:

$$\theta_{k+1} = \theta_k - \alpha_k \nabla_\theta L(\theta_k) \tag{67}$$

where α_k is the *learning rate* at the k^{th} step. and, gradient descent updates parameter by below rule:

$$\theta_{k+1}^{GD} = \theta_k + \alpha_k \frac{1}{n} D(\theta_k)^\top R_k, \tag{68}$$

The fitted values from gradient descent satisfy

$$\widehat{g(\theta_{k+1})} = g(\theta_k) + \frac{\alpha_k}{n} D_k D_k^\top R_k \tag{69}$$
$$= g(\theta_k) + \frac{\alpha_k}{n} K_k R_k \tag{70}$$

To compare the two algorithms, let us write D_k in its singular value decomposition form

$$D_k = U_k \Sigma_k V_k^\top \tag{71}$$

Where U_k is an $n \times n$ orthogonal matrix, Σ_k is an $n \times p$ diagonal matrix with n non-zero diagonal entries $(\sigma_1, \sigma_2, \ldots, \sigma_n)$ and V_k is a $p \times p$ orthogonal matrix. Then, the update of GD is

$$\begin{aligned} \theta_{k+1}^{GD} &= \theta_k + \frac{\alpha_k}{n} D_k^\top R_k \\ &= \theta_k + \frac{\alpha_k}{n} V_k \Sigma_k^\top U_k^\top R_k \end{aligned} \tag{72}$$

The update of regression descent at k^{th} step is

$$\theta_{k+1}^{RD} = \theta_k + D_k^\top \hat{\gamma}_k \tag{73}$$

Algorithm 2: Regression Descent – Extended Version

Input: Initial parameters $\theta_0 \in \mathbb{R}^p$, data $\{(x^{(i)}, y^{(i)})\}_{i=1}^n$, max iterations K
Input: Number of output classes $C \geq 1$; `// C = 1 for regression or binary classification`
Input: Mode: `batch_mode` $\in \{$`full`, `mini`$\}$, batch size m (if mini-batch)
Input: Regularization: `reg_mode` $\in \{$`fixed`, `adaptive`$\}$, initial $c_0 > 0$
Input: Adaptive parameters (if applicable): $\rho \in (0,1)$, $\eta_{\text{inc}} > 1$, $\eta_{\text{dec}} \in (0,1)$
Output: Trained parameters θ_K

for $k = 0, 1, \ldots, K-1$ **do**

`/* Step 1: Select data batch */`
if *batch_mode = full* **then**
$S_k \leftarrow (X, Y)$; `// full dataset`
$n_k \leftarrow n$;
else
$S_k \leftarrow$ `GetMiniBatch`(X, Y, m, k) ; `// non-overlapping batches, all` n `samples covered per epoch`
$n_k \leftarrow |S_k|$;

`/* Step 2: Compute predictions and residuals */`
$g_k \leftarrow [g(x^{(i)}; \theta_k)]_{x^{(i)} \in S_k} \in \mathbb{R}^{n_k \times C}$;
$R_k \leftarrow [y^{(i)}]_{x^{(i)} \in S_k} - g_k \in \mathbb{R}^{n_k \times C}$;
$R_k^{(\cdot j)} \leftarrow j$-th column of R_k, $\quad (R_k^{(\cdot j)} \in \mathbb{R}^{n_k})$, $\quad j = 1, \ldots, C$; `// per-class residuals`

`/* Step 3: Compute per-class Jacobians */`
$D_k^{(j)} \leftarrow \left[\frac{\partial g_j(x^{(i)}; \theta_k)}{\partial \theta}\right]_{x^{(i)} \in S_k} \in \mathbb{R}^{n_k \times p}$, $\quad j = 1, \ldots, C$; `// reduces to` j`=1 when` $C = 1$

`/* Step 4: Form per-class Gram matrix and solve linear system */`
$K_k^{(j)} \leftarrow D_k^{(j)} D_k^{(j)\top} \in \mathbb{R}^{n_k \times n_k}$, $\quad j = 1, \ldots, C$;
$\hat{\gamma}_k^{(j)} \leftarrow (K_k^{(j)} + n_k C c_k I)^{-1} R_k^{(\cdot j)}$, $\quad j = 1, \ldots, C$; `// one regularized system per class`

`/* Step 5: Compute parameter update */`
$\beta_k \leftarrow \frac{1}{C} \sum_{j=1}^{C} D_k^{(j)\top} \hat{\gamma}_k^{(j)} \in \mathbb{R}^p$; `// average over classes; reduces to` j`=1 when` $C = 1$

`/* Step 6: Adaptive regularization (optional) */`
if *reg_mode = adaptive* **then**
$R_{S_k} \leftarrow Y_{S_k} - g(X_{S_k}, \theta_k)$; `// residuals for current batch;` $\in \mathbb{R}^{n_k \times C}$ `for multiclass`
$\theta_{k+1} \leftarrow \theta_k + \beta_k$; `// trial update`
$R_{S_{k+1}}^{\text{trial}} \leftarrow Y_{S_{k+1}} - g(X_{S_{k+1}}, \theta_{k+1})$; `// trial residuals on next batch;` $\in \mathbb{R}^{n_{k+1} \times C}$
$\rho_k \leftarrow \dfrac{\|R_{S_{k+1}}^{\text{trial}}\|_F^2}{\|R_{S_k}\|_F^2}$; `// residual reduction ratio; Frobenius norm sums over all observations and classes`
$\alpha^* \leftarrow \dfrac{-\langle R_{S_k},\, R_{S_{k+1}}^{\text{trial}} - R_{S_k} \rangle_F}{\|R_{S_{k+1}}^{\text{trial}} - R_{S_k}\|_F^2}$; `// interpolation parameter; dot product extended to matrices via Frobenius inner product`
if $\rho_k < 0.8$ ***and*** $\alpha^* > 0.9$ **then**
$c_{k+1} \leftarrow c_k / \tau$; `// good model: decrease regularization`
else if $\rho_k > 0.9$ ***or*** $\alpha^* < 0.5$ **then**
$c_{k+1} \leftarrow c_k \cdot \nu$; `// poor model: increase regularization`
else
$c_{k+1} \leftarrow c_k$; `// no change`
else
$\theta_{k+1} \leftarrow \theta_k + \beta_k$; `// fixed regularization: always accept`
$c_{k+1} \leftarrow c_k$;

return θ_K

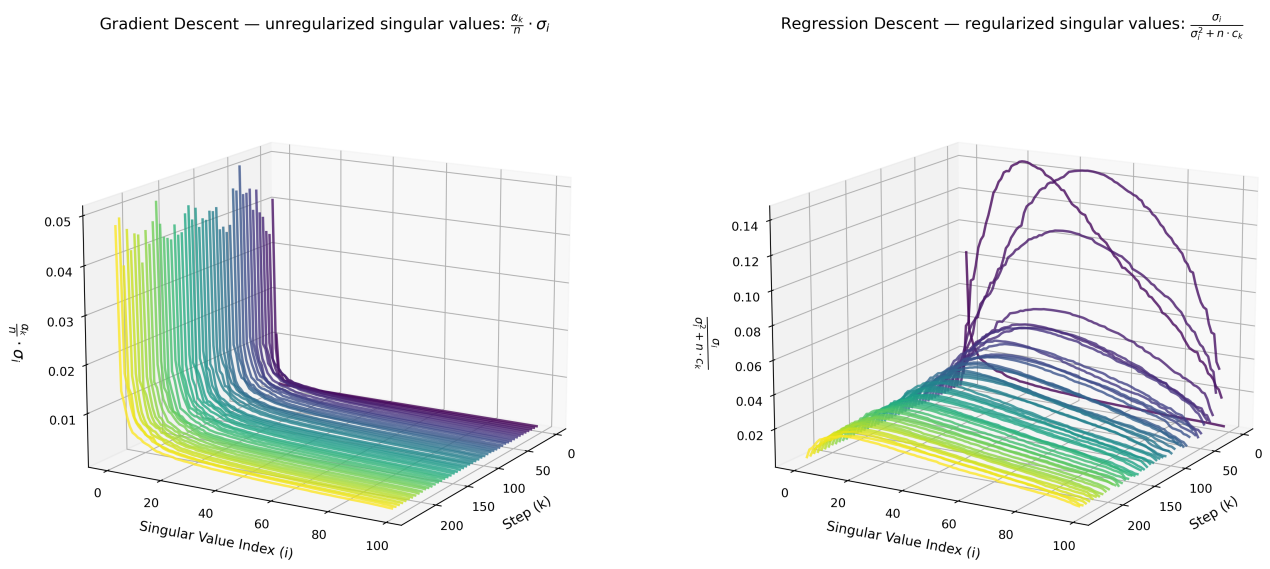


Figure 5: Singular value spectra for the deep network during training.

Then

$$D_k^\top \hat{\gamma_k} = D_k^\top (K_k + nc_k I_n)^{-1} R_k \tag{74}$$
$$= D_k^\top (K_k + nc_k I_n)^{-1} R_k \tag{75}$$
$$= V_k \Sigma_k^\top (\Sigma_k \Sigma_k^\top + nc_k I_n)^{-1} U_k^\top R_k \tag{76}$$
$$= V_k \tilde{\Sigma_k}^\top U_k^\top R_k \tag{77}$$

Where $\tilde{\Sigma}_k$ is an $n \times p$ matrix given by

$$\tilde{\Sigma}_k = \left[\text{diag}\left(\frac{\sigma_1}{\sigma_1^2 + nc_k}, \ldots, \frac{\sigma_n}{\sigma_n^2 + nc_k} \right) \quad \mathbf{0}_{n \times (p-n)} \right]$$

Before comparing equation 72 with equation 77 directly, first let us dig deeper into the structure of $V_k \Sigma_k^\top U_k^\top R_k$. We can write the update of gradient descent as

$$\frac{\alpha_k}{n} D_k^\top R_k = \frac{\alpha_k}{n} \sum_{i=1}^{n} \sigma_i (u_i^\top R_k) v_i \tag{78}$$

This can be interpreted as a weighted sum of the right singular vectors v_i, where each weight is the product of: how much the residual vector R_k aligns with the left singular vector u_i, i.e., $u_i^\top R_k$, and the corresponding singular value λ_i, multiplied by $\frac{\alpha_k}{n}$. On the other hand, the update of Regression Descent can be decomposed as follows:

$$V_k \tilde{\Sigma_k}^\top U_k^\top R_k = \sum_{i=1}^{n} \frac{\sigma_i}{\sigma_i^2 + nc_k} (u_i^\top R_k) v_i \tag{79}$$

By carefully evaluating equations 78 and 79, we make the following observations:

The update in gradient descent is dominated by the largest singular values of D_k, and it can struggle if these large singular values correspond to less important directions for reducing the residual.

On the other hand, RD normalizes the spectrum through adaptive weighting $\frac{\sigma_i}{\sigma_i^2 + nc_k}$: large singular values are damped (contributing as $\propto 1/\sigma_i$), while small singular values are regularized (contributing as $\propto \sigma_i$). This spectral

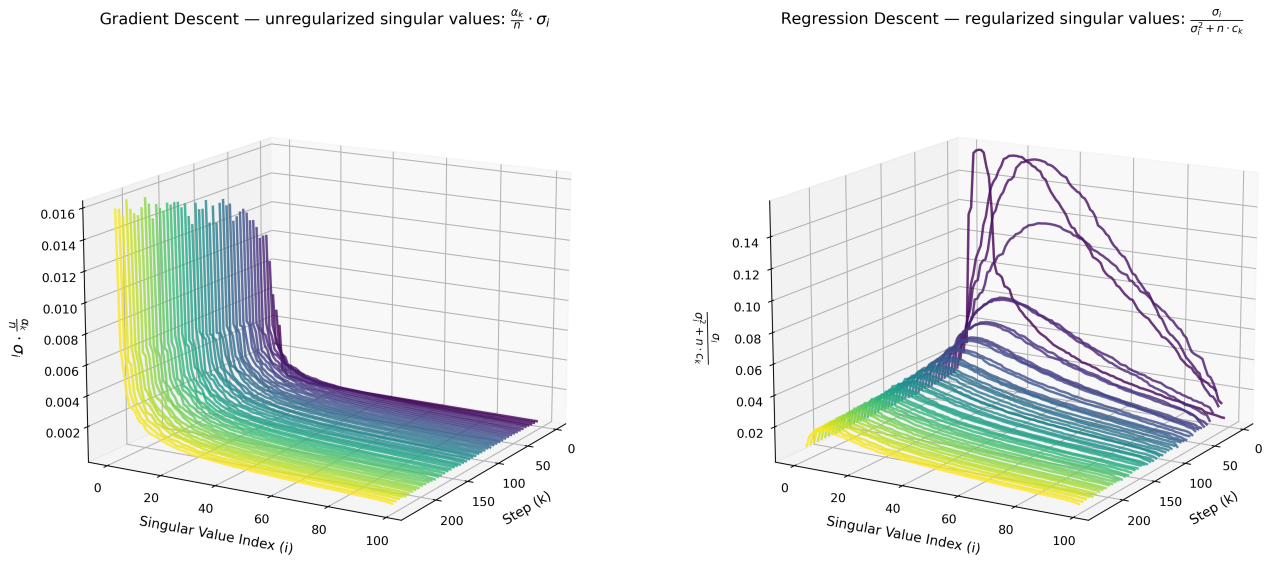
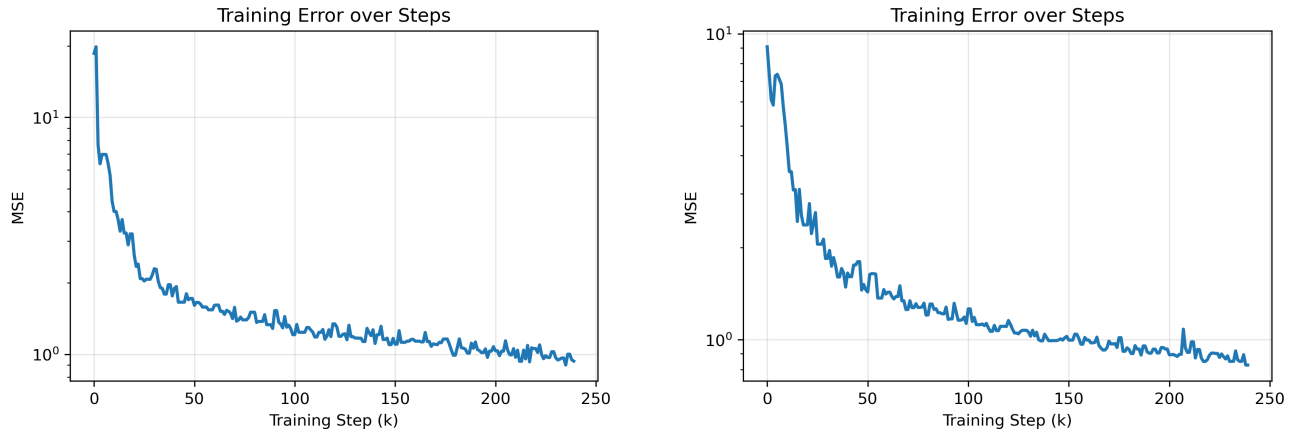


Figure 6: Singular value spectra for the wide network during training.

normalization allows RD to seek directions that balance fitting the residual R_k with numerical stability, making it robust to ill-conditioning.

Here we provide the singular value spectra of the two methods during training. We train the model using regression descent and, based on the Jacobian's SVD, plot the weighted singular value contributions given in equations 78 and 79 over the training steps, as shown in Figures 5 and 6. We use two network architectures on the Lorenz'96 dataset: a *deep network* with fully connected (FC) layers 10–32–32–32–32–16–8–1, and a *wide network* with FC layers 10–1032–32–1, both using ReLU activations without biases. The batch size is 99 (chosen because it divides the total number of observations n evenly, avoiding any confusion on the last batch; if the last batch contains fewer data points, the eigen spectrum is not directly comparable across training steps). We also provide the training error for reference in Figure 7. To calculate the eigen spectra of gradient descent, we used $\alpha_k = 0.01$ (a common choice of learning rate in (stochastic) gradient descent algorithms). For simplicity, in regression descent, c_k was initialized at 0.1 and increased at every epoch.



(a) Deep network

(b) Wide network

Figure 7: Training error curves

14 ADDITIONAL EXPERIMENTS

Here, we present two additional experiments that complement the main results: a CNN experiment on MNIST to further validate RD on convolutional architectures, and a large-scale ResNet-50 experiment to probe the scalability of RD.

We first evaluate RD on MNIST using a moderately deep convolutional architecture. The network processes 28×28 grayscale images through two convolutional blocks: the first with 8 filters (5×5 kernels) followed by 2×2 max pooling, and the second with 16 filters (5×5 kernels) followed by 2×2 max pooling. These reduce the spatial dimensions from $28 \times 28 \to 12 \times 12 \to 4 \times 4$, yielding a 256-dimensional feature vector. Two fully connected layers then map this to 64 units and finally to 10 class outputs, with ReLU activations after each convolutional layer and the first fully connected layer. We refer to this architecture as **CNN-2Conv**. Figure 8 shows the training performance of all four methods on CNN-2Conv; results are consistent with the main experiments.

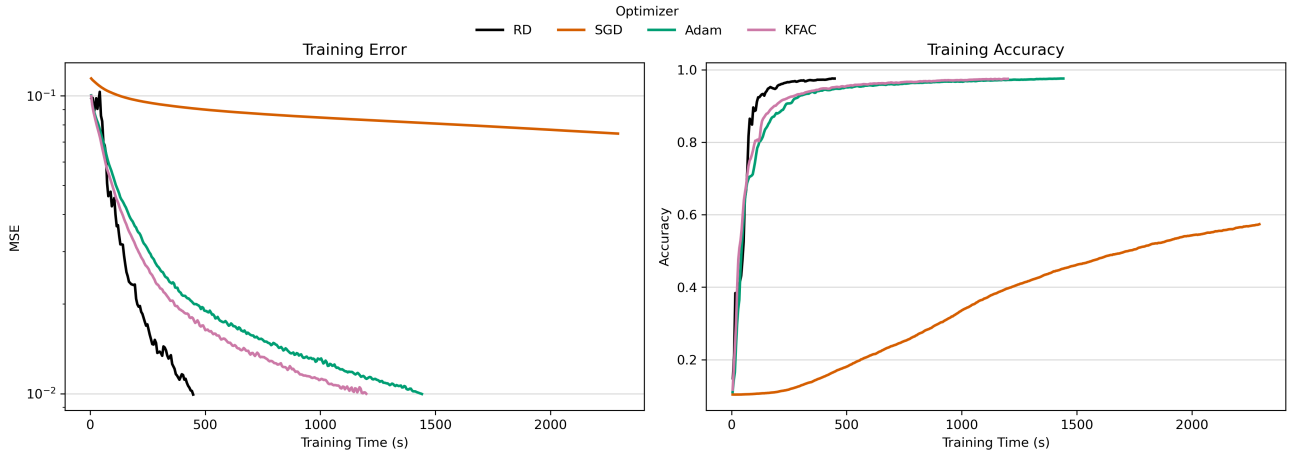


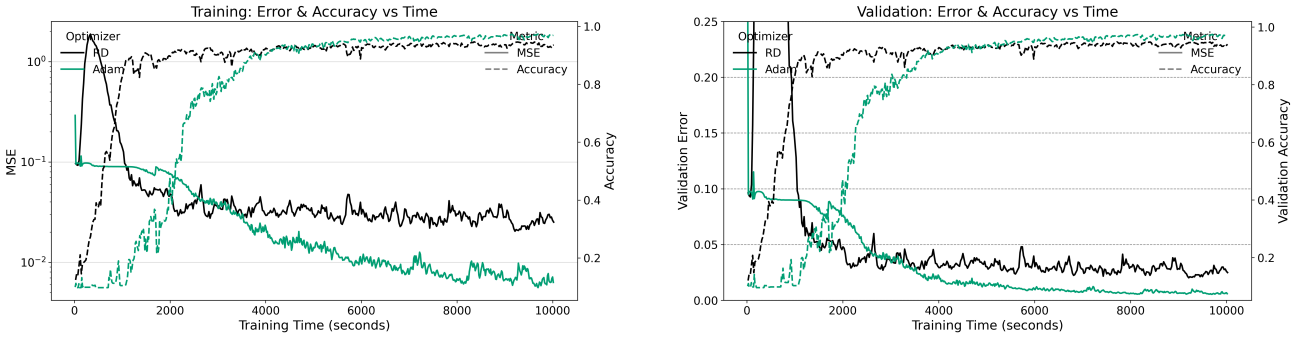
Figure 8: Training performance on the CNN-2Conv architecture on MNIST.

To evaluate whether RD scales to much larger architectures, we also trained a ResNet-50 on MNIST and compared it against Adam under identical experimental settings, with both methods run for 10,000 seconds. The architecture is a standard ResNet-50, adapted only at the input layer for MNIST's single-channel images. It begins with a 3×3 convolution with 64 channels, followed by four bottleneck stages with 3, 4, 6, and 3 blocks respectively. Each bottleneck expands the mid-channel dimensionality by a factor of 4, yielding progressively larger representations ($64 \to 256$, $128 \to 512$, $256 \to 1024$, $512 \to 2048$ channels), for a total of more than 23 million trainable parameters.

To make RD tractable at this scale, we avoided explicit Jacobian storage entirely, instead relying on on-the-fly matrix-vector products to compute the necessary quantities, which reduces memory overhead significantly. For RD, we used a fixed regularization parameter $c_k = 0.5$. The adaptive variant was not used here, as implementing it efficiently at this scale requires careful engineering to manage memory overhead during Jacobian inner-product computations; we leave this to future work. While the results are not yet competitive with Adam at this scale, the primary goal of this experiment is to demonstrate that RD can be applied to large architectures without fundamental scalability barriers. The training and validation curves are shown in Figure 9, with validation error plotted on a linear scale to make the separation between methods easier to interpret.

15 VALIDATION PERFORMANCE

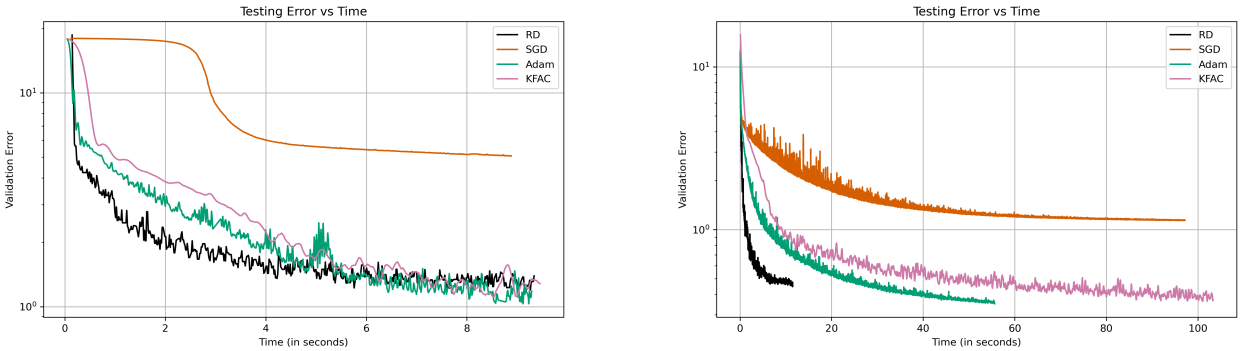
To complement the training results presented in Section 8, we report the corresponding validation performance for each architecture. All architectural details and experimental settings remain identical to those described in the main section.



(a) Training error & accuracy vs. time

(b) Validation error & accuracy vs. time

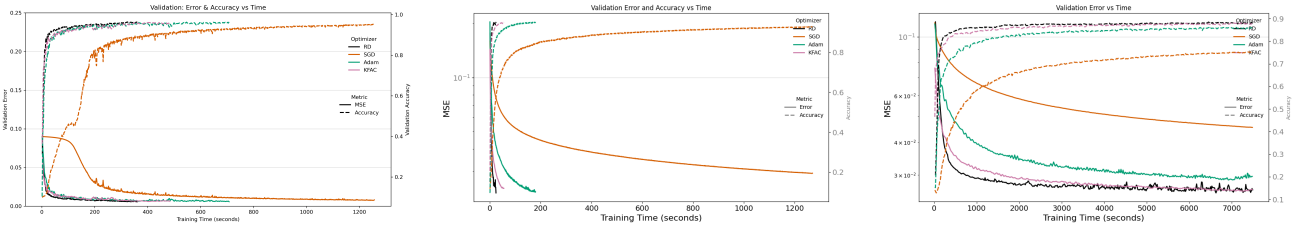
Figure 9: ResNet-50 trained on MNIST over 10,000 seconds of wall-clock time, comparing RD (with Jacobian-free matrix-vector products, $c_k = 0.5$) against Adam under identical experimental settings.



(a) SimpleDeepNN

(b) SimpleWideNN

Figure 10: Validation performance for L96 experiments (architectures as defined in Section 8). See Figures 1 and 2 for training performance.

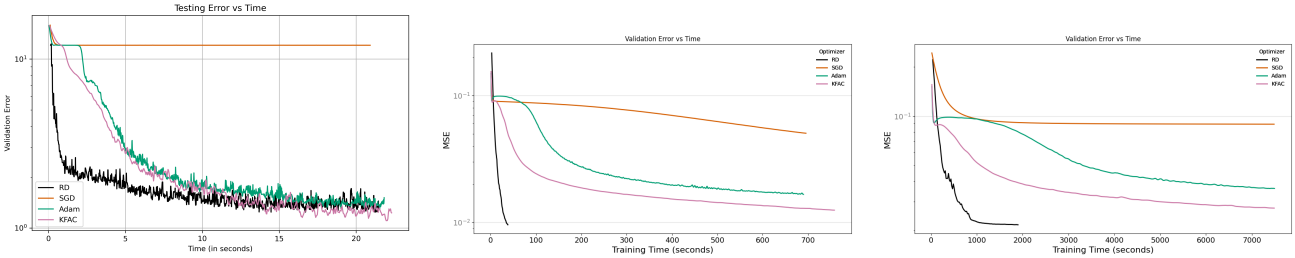


(a) FC on MNIST

(b) SmallCNN on MNIST

(c) TinyCNN on FMNIST

Figure 11: Validation performance for MNIST/FMNIST experiments (architectures as defined in Section 8). See Figure 3 for training performance.



(a) SimpleDeepNN on L96

(b) SmallCNN on MNIST

(c) SmallCNN on FMNIST

Figure 12: Validation performance for Sigmoid activation experiments (architectures as defined in Section 8). See Figure 4 for training performance.