A Globally Convergent Second-Order Optimization Method Utilizing the Diagonal Hessian in ReLU-Based Models

Insung Yun

Abstract

In this paper, we assume that in ReLU-based neural networks the negative absolute values of the diagonal entries of the Hessian are bounded by a small constant δ compared to a positive constant M (i.e. $-\delta \leq H_{ii} \leq M$, with $\delta \ll M$), and we propose a lightweight second-order optimization algorithm that exploits this property. The proposed method first constructs an approximate Hessian using only its diagonal entries and then ensures strict positive definiteness by adding a damping term λI. We mathematically prove that this approach significantly reduces the computational cost of typical second-order methods while guaranteeing global linear convergence. Furthermore, we apply the algorithm to multilayer perceptron architectures of various scales and demonstrate significant performance improvements compared to standard first-order optimizers.

Contents

1 Introduction

Deep learning optimization methods that leverage second-order information (the Hessian) offer theoretical advantages of fast convergence and high stability. However, conventional second-order methods incur high computational cost and are difficult to implement in practice, so most production-scale models rely on first-order algorithms such as Adam and Momentum. In this paper, we observe that despite their theoretical speed, existing second-order optimizers face scalability limits in large-scale parallel environments. To address this, we focus exclusively on a lightweight, diagonal-only second-order custom optimizer and compare its parallel efficiency directly against representative first-order methods (Adam, AdamW, AdaBelief).

Contributions.

- **Diagonal-only second-order update:** We propose a lightweight second-order method that secures strict positive definiteness using only the δ-bounded diagonal entries of the Hessian observed in ReLU/Leaky-ReLU networks.
- **Global convergence proof:** By combining transformation-function-based damping with momentum correction, we provide a theoretical guarantee of global linear convergence.
- **Empirical validation:** We experimentally demonstrate that our method consistently outperforms Adam-family optimizers in convergence speed and final performance across various MLP architectures.

Scope and Exclusions. This work concentrates on a lightweight second-order optimizer that uses only the diagonal of the Hessian matrix. Therefore, empirical comparisons with more complex second-order techniques such as Shampoo, K-FAC, and AdaHessian are outside the scope of this paper and will be addressed in future work.

Organization. The remainder of this paper is structured as follows:

- Section 2 reviews related work and highlights our contributions.
- Section 3 defines the problem and presents the mathematical assumptions.
- Section 4 details the proposed optimization algorithm and its formulation.
- Section 5 provides a proof of global linear convergence and discusses design intuitions such as flattening via transformation functions and escape from local minima.
- Section 6 describes the experimental setup, including models, hyperparameters, environment, datasets, and evaluation metrics.
- Section 7 presents the experimental results and analysis, covering performance comparisons, GPU utilization, and sensitivity studies.
- Section 8 concludes the paper, summarizes practical contributions and limitations, and outlines directions for future research.

Hyperparameter Note. The two main hyperparameters used in this work are diff (the fixed finite-difference step) and square (the parameter k for a $2k+1$-order transformation). We set

$$\text{diff} = 0.25, \quad \text{square} = 7 \quad (\text{i.e. } 2k+1 = 7 \Rightarrow k = 3).$$

As discussed in the experimental section, these values strike the best trade-off between generalization performance and convergence speed through empirical search.

2 Related Work

This work proposes an algorithm that retains the benefits of second-order optimization while minimizing computational cost and ensuring stable convergence even at saddle points and in nonconvex regions. We limit our comparisons to representative first-order optimizers: Adam, AdamW, and the recently proposed AdaBelief.

Adam [5] combines first-order momentum with an adaptive learning rate and is widely used for its practicality and convergence speed. **AdamW** [6] decouples weight decay from the regularization term to improve generalization. **AdaBelief** [12] replaces Adam's variance-based second moment estimate with the squared deviation, aiming for both fast convergence and better generalization.

Recent Trends in Lightweight Second-Order Methods To capture curvature information that is difficult for first-order optimizers to detect—while keeping costs low—recent work has focused on diagonal Hessian or low-rank approximation techniques. For example, HiZOO [11] and SOAA [9] significantly reduce memory and computation while preserving some second-order effects.

Comparison of Lightweight Second-Order Techniques Let n denote the number of hidden units per layer, so each weight matrix is $n \times n$ and the total parameter count is

$$N = n^2.$$

The computational complexities of various methods are summarized below:

- **Shampoo** [4], **K-FAC** [7]: Maintain two $n \times n$ preconditioning matrices per layer, requiring memory $O(n^2)$ and time $O(n^3)$ (i.e. $O(N^{3/2})$).
- **AdaHessian** [10]: Uses Hutchinson sampling for diagonal approximation, achieving memory $O(n^2)$ and time $O(k\,n^2) \approx O(n^2)$, but suffers variance issues when the number of probes k is small.
- **Our diagonal Hessian method**: Computes only the diagonal entries via batched tf.matmul/tf.tensordot without any inverse or sampling operations, thus memory $O(N) = O(n^2)$ and time $O(n^3)$ (per hidden-unit count n). Under XLA/JIT optimization, we experimentally verify reduced training time compared to Adam.

Key Innovations of This Paper:

- Instead of using the full Hessian or block approximations, we achieve drastic cost reduction and guarantee global convergence by utilizing **only the diagonal of the Hessian**.
- We apply the transformation

  $$g(x) = \big(f(x) - bl\big)^{2k+1}$$

  to flatten the local curvature of the loss function, and show that even a Newton update corrected solely by combining second-order information via a first-order momentum term can escape local traps. In our experiments, we set $k = 3$ (i.e. a 7th-power loss), which improved convergence stability in flattened regions and attainment of the global minimum.
- By applying **only first-order momentum** on top of this diagonal Hessian update, our method rapidly escapes saddle points and high-order nonlinear regions, addressing the saddle-point stagnation issue inherent in first-order methods [1].

3 Problem Definition and Mathematical Background

3.1 Problem Formulation

The training of the multilayer perceptron (MLP) considered in this paper is formulated as the following nonconvex optimization problem:

$$\min_{P} \ \mathcal{L}(P) = \min_{\{W^{(l)},b^{(l)}\}_{l=1}^{r}} \frac{1}{N}\sum_{i=1}^{N} \ell\big(f(X_i;P),\, Y_i\big),$$

where $P = \{P^{(l)}\}_{l=1}^{r}$ with $P^{(l)} = [W^{(l)} \quad b^{(l)}]$ gathers the weights and biases of layer l, $f(X;P)$ denotes the MLP with Leaky ReLU activation function f_r and softmax output, and $\ell(\hat{y}, y) = -\sum_k y_k \log \hat{y}_k$ is the categorical cross-entropy loss.

Baseline Shift (ε-Fixed) Strategy

At each epoch, we update

$$bl \leftarrow f(x_t) - \varepsilon, \quad \Longrightarrow \quad \varepsilon = f(x_t) - bl \quad (\varepsilon \ll 1),$$

thereby maintaining the flattening effect of the transformation $(f(x) - bl)^{2k+1}$.

3.2 Model Architecture and Notation

Throughout this paper, we use the following notation:

- $X^{(1)} \in \mathbb{R}^{n_1 \times B}$: input-layer activations (batch size B)
- $Z^{(l)} = W^{(l)}X^{(l)} + b^{(l)}$, $X^{(l+1)} = f_r(Z^{(l)})$ for $l = 1, \ldots, r-1$, where f_r is Leaky ReLU
- $\hat{Y} = f(Z^{(r)}) \in \mathbb{R}^{n_r \times B}$: softmax output
- $\widetilde{X}^{(l)} = [X^{(l)}; 1] \in \mathbb{R}^{(n_{l-1}+1)\times B}$: bias-augmented input
- B: batch size
- M: number of epochs
- S: steps per epoch
- r: number of weight layers
- n_l: number of neurons in layer l, and $n := \max_l n_l$

3.3 Key Assumptions

To guarantee global convergence of our second-order method, we make the following assumptions.

Assumption 1 ($\boldsymbol{\delta}$-Bounded Negativity of Hessian Diagonal). For any parameter P, the diagonal entries of the loss Hessian satisfy

$$-\delta \ \le \ H_{ii}(P) \ \le \ M, \qquad 0 < \delta \ll M.$$

That is, the absolute value of any negative diagonal element is strictly bounded by δ.

This assumption is empirically validated by a pilot test on a 7-layer Leaky-ReLU MLP (7 hidden layers of 64 units each, batch size 128) trained on MNIST. We repeated training for 150 epochs with fixed random seeds 42, 123, and 2025, and measured, over all epochs, the minimum diagonal Hessian entry δ, the maximum M, and their ratio δ/M. The results are shown in Table 1.

Table 1: Pilot test results: global δ (min H_{ii}), M (max H_{ii}), δ/M, and validation loss for the 7-layer Leaky-ReLU MLP (64×7 units), measured over all epochs.

Seed	δ (min H_{ii})	M (max H_{ii})	δ/M	Val Loss
42	−3.2263	92.354	0.0349	0.1039
123	−5.0300	39.097	0.1287	0.0998
2025	−18.143	282.430	0.0642	0.0943
Overall max			**0.1287**	

As the table shows, the maximum δ/M across the three runs is only about 0.13, and in most cases observed below 0.1. This strongly supports our $\delta \ll M$ assumption from an empirical standpoint. Sagun et al. reported that most Hessian eigenvalues concentrate near zero with only a few outliers protruding from the bulk [8], and Ghorbani et al. confirmed that this bulk concentration phenomenon persists even in large deep networks [3]. Therefore, our $\delta \ll M$ assumption is supported both empirically and by the literature.

Remark (Bound on Diagonal Entries via Rayleigh Quotient). For any symmetric matrix H and standard basis vector e_i, the Rayleigh quotient gives

$$\lambda_{\min}(H) \leq \frac{e_i^\top H\, e_i}{e_i^\top e_i} = H_{ii} \leq \lambda_{\max}(H).$$

Under Assumption 1, $\lambda_{\min}(H) \geq -\delta$ and $\lambda_{\max}(H) \leq M$, which implies $-\delta \leq H_{ii} \leq M$.

Assumption 2 (Local Gradient and Hessian Bound). Assuming $\|W^{(l)}\| \leq R$ for each layer and $\|x\| \leq B$ for inputs, the loss $\mathcal{L}(P)$ satisfies

$$\|\nabla \mathcal{L}(x)\| \leq G_{\text{loc}}, \quad \nabla^2 \mathcal{L}(x) \preceq M_{\text{loc}}\, I,$$

where G_{loc} and M_{loc} are derived in Sections IV–V of [2].

3.4 Research Questions and Contributions

This paper proposes a lightweight second-order optimizer using only the Hessian diagonal and addresses the following questions:

- **Convergence improvement:** How much does the dynamic learning rate based on Hessian diagonal improve convergence speed compared to first-order optimizers (Adam, AdamW, AdaBelief)?
- **Depth scalability:** How do computational cost and convergence stability change as the number of hidden layers r increases?
- **Statistical significance:** Does the proposed method show statistically significant performance differences compared to existing optimizers?

In Section 4, we detail the second-derivative-based learning rate computation and the algorithmic structure.

4 Proposed Method

Assumption 1 guarantees that at every iteration, the diagonal entries of the loss function's Hessian, H_{ii}, satisfy

$$-\delta \;\le\; H_{ii} \;\le\; M, \qquad 0 < \delta \ll M.$$

In other words, the absolute value of any negative diagonal entry is strictly bounded above by the constant δ.

Hessian Positivization via Transformation Function By defining the transformation function

$$g(x) = \big(f(x) - bl\big)^{2k+1}$$

we obtain

$$\nabla^2 g(x) = 2k(2k+1)\,(f(x) - bl)^{2k-1}\,\nabla f(x)\,\nabla f(x)^\top \;+\; (2k+1)\,(f(x) - bl)^{2k}\,\nabla^2 f(x).$$

Hence, the diagonal approximation $\nabla^2 g(x)_{ii}$ satisfies

$$\nabla^2 g(x)_{ii} \;\ge\; -\big(2k+1\big)\,(f(x) - bl)^{2k}\,\delta,$$

and by adding a sufficiently large damping constant $\lambda \gg (2k+1)(f(x) - bl)^{2k}\,\delta$, we get

$$\widetilde{H}_{ii} = \big|\nabla^2 g(x)_{ii}\big| + \lambda \;\ge\; \lambda - (2k+1)(f(x) - bl)^{2k}\,\delta \;>\; 0.$$

Therefore, $\mathrm{diag}(\widetilde{H})$ is guaranteed to be strictly positive definite.

Spectral Error due to Absolute Value Transformation The error in the diagonal entries after taking the absolute value is

$$\big|\nabla^2 g(x)_{ii}\big| - \nabla^2 g(x)_{ii} \;\le\; 2\,(2k+1)\,(f(x) - bl)^{2k}\,\delta.$$

Let $\varepsilon = f(x) - bl$. Then the spectral norm error incurred by the absolute value transformation is

$$2\,(2k+1)\,\varepsilon^{2k}\,\delta = \mathcal{O}\big(\varepsilon^{2k}\,\delta\big).$$

Under the assumption $\varepsilon \ll 1$, this error is $\mathcal{O}(\varepsilon^{2k}\,\delta)$, which is much smaller than δ, and thus never violates the δ–bounded negativity assumption of Assumption 1.

4.1 Algorithm and Model Architecture

- Input layer: $X^{(1)} \in \mathbb{R}^{n_1 \times \lambda}$
- l-th layer:
$$Z^{(l)} = W^{(l)}\,X^{(l)} + b^{(l)}, \quad X^{(l+1)} = f_r\big(Z^{(l)}\big)$$
where f_r is the Leaky ReLU activation.
- Output layer:
$$\hat{Y} = f\big(Z^{(r)}\big)$$
where f is the softmax activation.
- Parameter concatenation:
$$P^{(l)} = \big[\,W^{(l)} \;\; b^{(l)}\big], \quad \widetilde{X}^{(l)} = \begin{bmatrix} X^{(l)} \\ 1 \end{bmatrix}.$$
- Overall batch size: B

4.2 Second-Order Derivative-Based Learning Rate Computation

1. Hessian Diagonal We define the second derivative of $\mathcal{L}$ with respect to parameter $P_{ab}^{(l)}$ as

$$H_{ab}^{(l)} = \frac{1}{B} \sum_{k} \frac{\partial^2 \mathcal{L}}{\partial (P_{ab}^{(l)})^2}.$$

2. Expansion of Second-Derivative Terms

$$\frac{\partial^2 \mathcal{L}}{\partial (P_{ab}^{(r)})^2} = -\frac{1}{B} \sum_{k} \frac{\partial \hat{y}_{ak}}{\partial Z_{ak}^{(r)}} (\widetilde{X}_{bk}^{(r)})^2, \tag{1}$$

$$\begin{aligned}\frac{\partial^2 \mathcal{L}}{\partial (P_{ab}^{(r-1)})^2} &= \frac{1}{B} \sum_{i,k} (W^{(r)\,T})_{ai}\, \hat{y}_{ik}\, W_{ia}^{(r)}\, [f_r'(Z_{ak}^{(r-1)})]^2\, (\widetilde{X}_{bk}^{(r-1)})^2 \\ &\quad - \frac{1}{B} \sum_{i,j,k} (W^{(r)\,T})_{ai}\, \hat{y}_{ik}\, (W^{(r)\,T})_{aj}\, \hat{y}_{jk}\, [f_r'(Z_{ak}^{(r-1)})]^2\, (\widetilde{X}_{bk}^{(r-1)})^2,\end{aligned} \tag{2}$$

$$\begin{aligned}\frac{\partial^2 \mathcal{L}}{\partial (P_{ab}^{(r-2)})^2} &= \frac{1}{B} \sum_{i,s,j,k} (W^{(r-1)\,T})_{as} (W^{(r-1)\,T})_{aj}\, [(W^{(r)\,T})_{ji} (W^{(r)\,T})_{si}\, \hat{y}_{ik}] \\ &\quad \times f_r'(Z_{sk}^{(r-1)})\, f_r'(Z_{jk}^{(r-1)})\, [f_r'(Z_{ak}^{(r-2)})]^2\, (\widetilde{X}_{bk}^{(r-2)})^2 \\ &\quad - \frac{1}{B} \sum_{i,s,k} \Big[(W^{(r-1)\,T})_{as}\, [(W^{(r)\,T})_{si}\, \hat{y}_{ik}\, f_r'(Z_{sk}^{(r-1)})]\, f_r'(Z_{ak}^{(r-2)}) \Big]^2\, (\widetilde{X}_{bk}^{(r-2)})^2\end{aligned} \tag{3}$$

3. Generalization for the $r-m$ Layer

$$\begin{aligned}\frac{\partial^2 \mathcal{L}}{\partial (P_{ab}^{(r-m)})^2} &= \frac{1}{B} \sum_{s,j,k} (W^{(r-m+1)\,T})_{as} (W^{(r-m+1)\,T})_{aj}\, D_{sjk}^{(r-m)}\, [f_r'(Z_{ak}^{(r-m)})]^2\, (\widetilde{X}_{bk}^{(r-m)})^2 \\ &\quad - \frac{1}{B} \sum_{k} [J_{ak}^{(r-m)}]^2\, (\widetilde{X}_{bk}^{(r-m)})^2,\end{aligned} \tag{4}$$

$$\begin{aligned}\frac{\partial^2 \mathcal{L}}{\partial (P_{ab}^{(r-m-1)})^2} &= \frac{1}{B} \sum_{\alpha,\beta,s,j,k} (W^{(r-m)\,T})_{a\alpha} (W^{(r-m)\,T})_{a\beta}\, (W^{(r-m+1)\,T})_{\alpha s} (W^{(r-m+1)\,T})_{\beta j}\, D_{sjk}^{(r-m)} \\ &\quad \times f_r'(Z_{\alpha k}^{(r-m)})\, f_r'(Z_{\beta k}^{(r-m)})\, [f_r'(Z_{ak}^{(r-m-1)})]^2\, (\widetilde{X}_{bk}^{(r-m-1)})^2 \\ &\quad - \frac{1}{B} \sum_{s,k} \big[(W^{(r-m)\,T})_{as}\, J_{sk}^{(r-m)}\, f_r'(Z_{ak}^{(r-m-1)}) \big]^2\, (\widetilde{X}_{bk}^{(r-m-1)})^2.\end{aligned} \tag{5}$$

4. Definition of Intermediate Tensors

$$D_{sjk}^{(r-2)} = \sum_{h} (W^{(r)\,T})_{sh} (W^{(r)\,T})_{jh}\, \hat{y}_{hk}, \quad J_{ak}^{(r-1)} = \sum_{i} (W^{(r)\,T})_{ai}\, \hat{y}_{ik}\, f_r'(Z_{ak}^{(r-1)}).$$

$$D_{\alpha\beta k}^{(r-m-1)} = \sum_{s,j} (W^{(r-m+1)\,T})_{\alpha s} (W^{(r-m+1)\,T})_{\beta j}\, D_{sjk}^{(r-m)}\, f_r'(Z_{\alpha k}^{(r-m)})\, f_r'(Z_{\beta k}^{(r-m)}),$$

$$J_{ak}^{(r-m-1)} = \sum_{s} (W^{(r-m)\,T})_{as}\, J_{sk}^{(r-m)}\, f_r'(Z_{ak}^{(r-m-1)}).$$

5. Definition of `_Hsolve` For an input tensor $T \in \mathbb{R}^{n_l \times B}$, extended input $\widetilde{X} \in \mathbb{R}^{(n_{l-1}+1)\times B}$, first-order gradient matrix $dP \in \mathbb{R}^{n_l \times (n_{l-1}+1)}$, batch size B, and coefficients $\alpha_{\text{out}}, \alpha_H$, we compute

$$H = \frac{1}{B}\, T\, \big(\widetilde{X}^{\circ 2}\big)^T, \quad \text{outer\_dP} = dP^{\circ 2},$$

$$\widetilde{H} = \big|\alpha_{\text{out}}\, \text{outer\_dP} + \alpha_H\, H\big| + \lambda, \quad \lambda = 10^{-2}\ \text{(stabilization constant)}$$

$$L = \frac{\alpha_H\, dP}{\widetilde{H}}.$$

Remark (Rationale for Setting Damping Constant λ). Preliminary experiments were conducted over $\lambda \in \{10^{-4}, 10^{-3}, 10^{-2}\}$, and $\lambda = 10^{-2}$ showed the most consistent generalization performance. Consequently, this value was fixed in all subsequent experiments.

6. Weight-Specific Learning Rate

$$L_{ab} = \frac{dP_{ab}\, \alpha_H}{\widetilde{H}_{ab}}.$$

4.3 Dynamic Learning Rate and Momentum Updates

Fundamental Equations By introducing loss-based damping and a global scale η_0, we define

$$\text{damp}(x) = \text{shifted\_normal}(x), \quad \eta_0 = 2 \times 10^{-3}, \quad \text{lr} = \frac{\eta_0\, \text{damp}(\mathcal{L})}{\text{diff}^{(\text{square}-1)}}.$$

Definition of Damping Function

$$\bar{x} = \min\big(\max(x, 0), 2\big), \quad \text{damp}(x) = (M^{'} - \text{base})\, \exp\Big(-\frac{(\bar{x} - \mu)^2}{2\sigma^2}\Big) + \text{base},$$

where $\mu = 0.7$, $M^{'} = 0.125$, $\sigma = 0.575$, and base $= 0.01$. Each parameter was set experimentally: μ denotes the peak location, $M^{'}$ the maximum output, σ the distribution width, and base the convergence value as $x \to \infty$.

Learning Rate Separation The learning rate tensor $L \in \mathbb{R}^{n_l \times (n_{l-1}+1)}$ computed for each layer is separated as

$$d2W = L[:,\, 1 : n_{l-1}], \quad d2b = L[:,\, n_{l-1} + 1 : n_{l-1} + 2],$$

to match the corresponding momentum variables.

Momentum Update

$$m_W \leftarrow \beta_1\, m_W + (1 - \beta_1)\, d2W, \quad \hat{m}_W = \frac{m_W}{1 - \beta_1^t},$$

$$m_b \leftarrow \beta_1\, m_b + (1 - \beta_1)\, d2b, \quad \hat{m}_b = \frac{m_b}{1 - \beta_1^t},$$

$$W \leftarrow W - \text{lr}\, \hat{m}_W, \quad b \leftarrow b - \text{lr}\, \hat{m}_b.$$

In the following sections, we analyze the convergence and performance of the proposed optimizer based on these settings.

Table 2: Key Hyperparameters

Symbol	Description	Default Value or Range
M	Number of epochs (Epochs)	250
B	Batch size (Batch size)	128
S	Steps per epoch (#steps per epoch)	$\lceil N/B \rceil$
diff	Fixed difference ($f(x) - bl = 0.25$)	0.25
square	Exponent order coefficient	7
α_{out}	Difference-square scaling coefficient (diff-based)	$\text{square}\left(\text{square}-1\right)\text{diff}^{\,\text{square}-2}$
α_H	Hessian coefficient	$\text{square}\,\text{diff}^{\,\text{square}-1}$
β_1	First-order momentum coefficient	0.25
η_0	Global learning rate scale	2×10^{-3}
λ	Stabilization constant	10^{-2}

Table 3: Key Hyperparameters and Rationale

Symbol	Value	Rationale for Setting
diff	0.25	Minimized loss oscillation and optimized convergence speed during grid search over [0.1–0.5]
square	7	Best trade-off between flattening effect and fast convergence in tests over orders 5–9
η_0	2×10^{-3}	Exhibited the fastest initial convergence after experiments over initial learning rates in the range [10^{-4}–10^{-2}]

4.4 Overall Algorithm Flow

Algorithm 1 Second-Order Derivative-Based Optimizer (Reflecting TensorFlow Implementation Structure)

Require: dataset 'dataset', parameters $\{W^{(l)}, b^{(l)}\}$, batch size B, number of epochs M, steps per epoch S

▷ *Note: This pseudocode uses einsum notation for clarity; the actual implementation uses `tf.tensordot`.*

for epoch $t = 1, \ldots, M$ **do**
  for each $(X_{\text{batch}}, Y_{\text{batch}})$ in dataset.take(S) **do**
    ▷ — Forward propagation and error computation —
    compute $\{Z^{(l)}, X^{(l+1)}\}$ and $\hat{Y} = f(Z^{(r)})$
    $J \leftarrow \hat{Y} - Y_{\text{batch}}$
    Let $W^T = W^{(r)\,T}$. Then
    $D \leftarrow$ einsum('is,js,sk', W^T, W^T, $\hat{Y}$)
    $Je \leftarrow (W^T \hat{Y}) \odot f_r'(Z^{(r-1)})$
    compute α_{out}, α_H from 'square', 'diff'
    for $l = r, r-1, \ldots, 1$ **do**
      ▷ — First-order gradient computation —
      $dW \leftarrow \frac{1}{B} J\, X^{(l)\,T}, \quad db \leftarrow \frac{1}{B} \sum J$
      $dP \leftarrow [\, dW \;\; db]$
      if $l > 1$ **then**
        ▷ — Backpropagation: apply activation derivative —
        $J \leftarrow W^{(l)\,T} J \odot f_r'(Z^{(l-1)})$
      end if
      ▷ — Branch computation of Hessian term T —
      if $l = r$ **then**
        $T \leftarrow \hat{Y} - \hat{Y}^2$
      else if $l = r - 1$ **then**
        $T \leftarrow ((W^{(l+1)\,T})^{\circ 2}\, \hat{Y} - (W^{(l+1)\,T} \hat{Y})^{\circ 2}) \odot (f_r'(Z^{(l)})^{\circ 2})$
      else
        Let $W^T = W^{(l+1)\,T}$, $\delta = f_r'(Z^{(l)})$.
        Let $E =$ einsum('as,aj,sjk->ak', W^T, W^T, D).
        Then $T \leftarrow (E - (W^T Je)^{\circ 2}) \odot \delta^{\circ 2}$.
        $D \leftarrow$ einsum('ij,kl,jlm,im,km->ikm', $W^T, W^T, D, \delta, \delta$).
        $Je \leftarrow (W^T Je) \odot \delta$.
      end if
      ▷ — Hessian-based learning rate computation —
      $L \leftarrow \_Hsolve(T, [X^{(l)}; 1], dP, B, \alpha_{\text{out}}, \alpha_H)$ ▷ (see §4.2)
      split L into $d2W$ (first n_{l-1} columns) and $d2b$ (last column)
      ▷ — Loss-based damping & lr computation —
      $damp \leftarrow$ shifted_normal($\mathcal{L}$)
      $lr \leftarrow \eta_0 \frac{damp}{\text{diff}^{(\text{square}-1)}}$
      ▷ — Momentum update —
      $m_W \leftarrow \beta_1 m_W + (1 - \beta_1)\, d2W, \quad \hat{m}_W \leftarrow \frac{m_W}{1-\beta_1^t}$
      $m_b \leftarrow \beta_1 m_b + (1 - \beta_1)\, d2b, \quad \hat{m}_b \leftarrow \frac{m_b}{1-\beta_1^t}$
      ▷ — Parameter update —
      $W^{(l)} \leftarrow W^{(l)} - lr\, \hat{m}_W, \quad b^{(l)} \leftarrow b^{(l)} - lr\, \hat{m}_b$
    end for
  end for
end for

Time Complexity Analysis

(This analysis is based on the actual TensorFlow function `_hessian_block_step`*, not the pseudocode in Section 4.4.)*

Considering the main computational costs per layer l for one batch in `_hessian_block_step`, we have the following. Here, r is the number of parameter blocks (weight matrices), n_l is the number of neurons in the l-th hidden layer, B is the batch size, and M is the number of epochs.

Notation. $n := \max_l n_l$ (maximum hidden-layer width)

1. _tensorOfH Operation ($l \leq r-2$ General Layer)

$$\begin{aligned}
&\text{tensordot} \ : \ O\big(B\, n_l\, n_{l-1}^2\big),\\
&\text{reduce_sum} \ : \ O\big(B\, n_l\, n_{l-1}\big),\\
&\text{matmul} \ : \ O\big(B\, n_l\, n_{l-1}\big),\\
&D \text{ update (batched matmul } \times 2 + \text{ broadcast mul)} : \ O\big(B\, n_l^2\, n_{l-1}\big),
\end{aligned}$$

Thus, the dominant term per layer is

$$O\big(B\, n_l\, n_{l-1}^2 + B\, n_l^2\, n_{l-1}\big) \ \approx \ O\big(B\, n^3\big).$$

2. _Hsolve Operation

$$H = \tfrac{1}{B}\, T\, (\widetilde{X}^{\circ 2})^T : O\big(n_l\, n_{l-1}\, B\big), \quad \text{element-wise ops} : O\big(n_l\, n_{l-1}\big), \quad L = \tfrac{\alpha_H\, dP}{\widetilde{H}} : O\big(n_l\, n_{l-1}\big).$$

Total per layer:

$$O\big(n_l\, n_{l-1}\, B\big) \ \approx \ O\big(n^2\, B\big).$$

3. First-Order Gradient & Momentum Update

$$\text{Backpropagation: } O\big(n_l\, n_{l-1}\, B\big), \quad \text{Parameter update: } O\big(n_l\, n_{l-1}\big) \ \approx \ O\big(n^2\, B\big).$$

Overall Batch Complexity Summing over all layers,

$$\sum_{l=1}^{r} O\big(B\, n^3\big) = O\big(r\, B\, n^3\big).$$

Multiplying by the number of epochs M gives a total training complexity of

$$O\big(M\, B\, r\, n^3\big).$$

Note: Adam Optimizer Assuming the number of parameters $P \approx \sum_l n_l n_{l-1}$,

$$O\big(M\, B\, P\big) \approx O\big(M\, B\, r\, n^2\big),$$

i.e., an n^2 dependency, which is significantly lower than the n^3 dependency of the proposed optimizer.

5 Theoretical Guarantees and Design Intuition

5.1 Theoretical Guarantees

In this section, we summarize the global convergence of the proposed optimization algorithm by leveraging the mathematical properties of the transformation-function-based Newton method. Detailed derivations are provided in Appendix A.

Under Assumption 1, for neural networks using ReLU-family activation functions, the diagonal entries of the Hessian of the loss function at any parameter x,

$$H_{ii} = [\nabla^2 f(x)]_{ii},$$

satisfy

$$-\delta \;\le\; H_{ii} \;\le\; M, \qquad 0 < \delta \ll M.$$

That is, the absolute value of any negative diagonal entry is strictly bounded above by the constant δ.

Under Assumption 2, assuming that each layer's weights satisfy $\|W^{(l)}\| \le R$ and that the input satisfies $\|x\| \le B$, it follows that

$$\|\nabla f(x)\| \;\le\; G_{\text{loc}} \;=\; G, \quad \nabla^2 f(x) \;\preceq\; M_{\text{loc}}\, I \;=\; M\, I.$$

Theorem 1 (Global Convergence Guarantee). For the loss function $f(x)$, define the transformation function

$$g(x) = \big(f(x) - bl\big)^{2k+1},$$

and construct the damped Newton update as follows:

$$x_{t+1} = x_t - \alpha \left[\nabla^2 g(x_t) + \lambda I\right]^{-1} \nabla g(x_t).$$

Preservation of Positive Semidefiniteness and Ensuring Damping. For the transformation function

$$g(x) = \big(f(x) - bl\big)^{2k+1},$$

we have

$$\nabla^2 g(x) = 2k(2k+1)\,(f(x) - bl)^{2k-1}\,\nabla f(x)\,\nabla f(x)^\top + (2k+1)\,(f(x) - bl)^{2k}\,\nabla^2 f(x).$$

Note. In this section, for notational simplicity we write $g(x) = (f(x) - bl)^{2k+1}$. In the actual algorithm, however, the baseline bl is updated at each iteration t via $bl_t = f(x_t) - \varepsilon$, so strictly speaking one should interpret it as $g_t(x) = (f(x) - bl_t)^{2k+1}$.

Ensuring Strict Positive Definiteness. Under Assumption 1, we have $\nabla^2 f(x) \succeq -\delta I$ and $\nabla f(x)\,\nabla f(x)^\top \succeq 0$. Hence,

$$\begin{aligned}\nabla^2 g(x) &= 2k(2k+1)\,(f(x) - bl)^{2k-1}\,\nabla f(x)\nabla f(x)^\top \\ &\quad +\; (2k+1)\,(f(x) - bl)^{2k}\,\nabla^2 f(x) \\ &\succeq -\,(2k+1)\,(f(x) - bl)^{2k}\,\delta\, I.\end{aligned}$$

By adding $\lambda \gg (2k+1)(f(x) - bl)^{2k}\,\delta$ with $\lambda > 0$, we obtain

$$\nabla^2 g(x) + \lambda I \;\succeq\; \left[\lambda - (2k+1)(f(x) - bl)^{2k}\,\delta\right] I \;=:\; \mu\, I \;\succ\; 0,$$

i.e., $\nabla^2 g(x) + \lambda I$ is always positive definite.

Definition of Upper Bound L_g and Damped Hessian Upper Bound. Let the loss-to-baseline difference be $\varepsilon = f(x) - bl$ with $\varepsilon > 0$. Then

$$\nabla^2 g(x) = 2k(2k+1)\,\varepsilon^{2k-1}\,\nabla f(x)\nabla f(x)^\top + (2k+1)\,\varepsilon^{2k}\,\nabla^2 f(x),$$

so by the above assumptions,

$$\nabla^2 g(x) \preceq 2k(2k+1)\,\varepsilon^{2k-1}G^2\,I + (2k+1)\,\varepsilon^{2k}M\,I.$$

Therefore, including the damping term λI,

$$\nabla^2 g(x) + \lambda I \preceq \Big[\,2k(2k+1)\,\varepsilon^{2k-1}G^2 + (2k+1)\,\varepsilon^{2k}M + \lambda\Big]\,I =: L_g\,I.$$

Moreover,

$$\nabla^2 g(x) + \lambda I \preceq L_g\,I, \quad 0 \prec \mu I \preceq \nabla^2 g(x) + \lambda I.$$

Hence, $\tilde{g}(x) = g(x) + \frac{\lambda}{2}\|x\|^2$ satisfies $\nabla^2\tilde{g}(x) \succeq \mu I$, implying μ-strong convexity. The minimizer x^* is unique and $\nabla\tilde{g}(x^*) = 0$. As shown in Appendix A, there exists a constant $\rho < 1$ such that for a suitable $\alpha \in \big(0, \frac{2\lambda^2}{L_g^2}\big)$ and $\lambda > 0$,

$$\tilde{g}(x_t) - \tilde{g}(x^*) = O(\rho^t) \quad \Longrightarrow \quad \|x_t - x^*\| \to 0 \quad (t \to \infty),$$

which establishes global linear convergence.

Range of α in This Case:

$$0 < \alpha < \frac{2\lambda^2}{L_g^2}.$$

Definition 5.1 (ε-Shift Preservation Assumption). In this proof, it is essential that $\varepsilon = f(x) - bl$ remain constant. To achieve this, we fix $\varepsilon > 0$ in advance and assume that at each iteration the shift constant bl is defined by

$$bl \leftarrow f(x) - \varepsilon.$$

Under this setting, $\varepsilon = f(x) - bl$ is preserved throughout the iterations, thereby reliably satisfying the assumption $\varepsilon > 0$ used in the convergence theorem.

5.2 Design Ideas and Intuition

Combining Second-Order Information in a Momentum Form This algorithm applies an exponential moving average to the diagonal Hessian entries D_t as

$$m_t = \beta\,m_{t-1} + (1-\beta)\,D_t, \quad \beta \in [0, 1),$$

thereby retaining a portion of past curvature information. By using the resulting m_t as a correction term in the Newton update, the algorithm maintains "inertia" even in flattened local minima regions, effectively escaping flat traps that would be difficult to exit using only the local gradient.

Flattening and Local Minima Escape Idea Introducing the transformation function

$$g(x) = (f(x) - bl)^{2k+1}$$

and maintaining the baseline bl such that $f(x) - bl = \varepsilon$ ($\varepsilon \ll 1$) yields

$$\nabla^2 g(x) = 2k(2k+1)\,\varepsilon^{2k-1}\nabla f(x)\,\nabla f(x)^\top + (2k+1)\,\varepsilon^{2k}\nabla^2 f(x).$$

In this expression, the first term is always positive semidefinite and of order $O(\varepsilon^{2k-1}\|\nabla f(x)\|^2)$ in the spectral norm. Thus, even in regions where the original function exhibits negative curvature, adding a sufficiently small $\lambda > 0$ guarantees $\nabla^2 g(x) + \lambda I \succ 0$.

Based on the above expression, by adjusting the baseline bl at each iteration to maintain

$$f(x) - bl = \varepsilon \quad (\varepsilon \text{ sufficiently small}),$$

the transformation function $g(x)$ has its gradient converging to zero near local minima, and the Hessian of the original function contracts at an $O(\varepsilon^{2k-1})$ scale. As a result, the Hessian becomes nearly zero, effectively flattening the curvature. In this state, the Newton update—with only the correction that integrates second-order information (the Hessian) in a momentum form—can effectively escape the local trap and move toward the global minimum. Even after reaching the global minimum, since the curvature remains flat and no lower undulations or other minima exist, the algorithm returns to the vicinity of the global minimum and ultimately converges stably.

6 Experimental Setup

6.1 Model Architecture

The MLP used in our experiments is configured as follows:

- **Number of hidden layers**: 7 or 10
- **Hidden units per layer**: 64
- **Activation function**: Leaky ReLU
- **Output layer**: Softmax

Hyperparameters We adopt the hyperparameters defined in Section 4.3, in particular batch size $B = 128$ and number of epochs $M = 250$.

6.2 Experimental Phases

1. **Vanilla MLP (no regularization):**
   - **Objective**: Evaluate optimizer performance on a pure MLP (7 or 10 layers) without any regularization
   - **Setup**: Hidden layers = 7, 10 / batch size = 128 / hidden units = 64 / Leaky ReLU / softmax
   - **Repetitions**: 7 random seeds

- **Metrics**: Training Loss, Validation Loss, Validation Accuracy, Macro F_1-Score, Training Time

2. **Comparison of Regularization Techniques:**
   - **Objective**: Assess the impact of regularization on the custom optimizer vs. baselines (Adam, AdamW, AdaBelief)
   - **Custom optimizer settings**: Dropout = 0.004, Label Smoothing = 0.025
   - **Baseline settings**: BatchNorm ON, Dropout = 0.004, Label Smoothing = 0.025
   - **Common**: Hidden layers = 7, 10 / batch size = 128 / hidden units = 64 / Leaky ReLU / softmax
   - **Repetitions**: 7 random seeds
   - **Metrics**: Training Loss, Validation Loss, Validation Accuracy, Macro F_1-Score, Training Time
3. **Hyperparameter Sensitivity Analysis:**
   - **Objective**: Measure validation loss sensitivity to Dropout–Label Smoothing combinations
   - **Fixed settings**:
     - BatchNorm: ON for baselines only (Adam, AdamW, AdaBelief)
     - Number of hidden layers: 7
     - Epochs: 100
     - Dataset: WineQuality
   - **Variable settings**:
     - Dropout: {0.002, 0.004, 0.008}
     - Label Smoothing α: {0.0125, 0.025, 0.05}
   - **Repetitions**: 3 random seeds per combination, record mean validation loss
   - **Metric**: Mean Validation Loss

6.3 Computational Environment

Hardware

- GPU: NVIDIA GeForce RTX 4060 (8 GB)
- CPU: AMD Ryzen 7 5700X 8-Core @ 3.40 GHz
- RAM: 32 GB DDR4-3200

Software

- OS: Ubuntu 22.04 LTS
- Python: 3.9
- CUDA / cuDNN: CUDA 12.1 (nvcc V12.1.105), cuDNN 9.9.0

- TensorFlow: 2.15.0 (XLA JIT compiler enabled)
- Key libraries:
  - tensorflow-addons 0.22.0
  - numpy 1.26.4
  - pandas 2.2.3
  - matplotlib 3.9.4
  - scipy 1.13.1
  - scikit-learn 1.6.1

6.4 Comparison Optimizers and Hyperparameters

We compare our second-derivative-based optimizer against three first-order methods: Adam, AdamW, and AdaBelief. All baselines use `learning_rate` = 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$. Additionally, AdamW and AdaBelief employ `weight_decay` = 1×10^{-4}.

6.5 Datasets and Evaluation Metrics

Datasets We use eight real-world datasets and four synthetic Gaussian datasets:

- **MNIST**: 60,000 train, 10,000 test; input dim 784; pixel values normalized to [0,1]
- **CIFAR-10**: 50,000 train, 10,000 test; input dim 32×32×3; pixel values normalized to [0,1]
- **CIFAR-100**: 50,000 train, 10,000 test; input dim 32×32×3; pixel values normalized to [0,1]
- **20 Newsgroups**: 18,846 documents; train/test split 80%/20%; TF–IDF vectors (2,000 dims)
- **Imbalance**: 30,000 samples, 4 classes (weights [0.7, 0.15, 0.1, 0.05]); train/test 80%/20%
- **WineQuality-Red**: 1,599 samples; 11 standardized features; 6-class one-hot labels; train/test 80%/20%
- **Fashion-MNIST**: 60,000 train, 10,000 test; input dim 784; pixel values normalized to [0,1]
- **HAR (UCI)**: 9-channel sensor data; train/test 80%/20%; data loaded from .npy files
- **Synthetic Gaussian**: 30,000 samples; 12 features; 8 classes; parameters: class_sep ∈ {0.5,1.0,2.0}, clusters ∈ {1,3,5}, flip_y ∈ {0,0.05}; train/test 80%/20%

These datasets span image, text, numerical, time-series, and synthetic domains to comprehensively evaluate the generalization and stability of our method. Source details (URL, authors, year, license) are listed in Appendix D.

Evaluation Metrics

- **Training Loss & Validation Loss**: Categorical cross-entropy at end of each epoch
- **Accuracy**: $\frac{\text{correctly classified samples}}{\text{total samples}}$
- **Macro F_1-Score**:

$$\text{F1}_{\text{macro}} = \frac{1}{C} \sum_{c=1}^{C} \frac{2 \, \text{Precision}_c \, \text{Recall}_c}{\text{Precision}_c + \text{Recall}_c}$$

- **Time to Convergence**: Total training time in seconds (measured via `time.perf_counter()`, stored in `buf['time_c']`)
- **Statistical Tests**:
  - Paired t-test ($\alpha = 0.05$)
  - Effect size (Cohen's d):
    - Large: $0.8 \leq |d| < 1.2$
    - Very large: $|d| \geq 1.2$

7 Experimental Results

Statistical Definitions

- $\Delta(\%)$: $\frac{\text{Custom} - \text{Baseline}}{\text{Baseline}} \times 100$ (Negative $\Rightarrow$ Custom advantage; Positive $\Rightarrow$ Custom disadvantage).
- **Mean**: Arithmetic mean of the Δ values over the 12 datasets (8 real + 4 synthetic).
- **Median**: Median of the same Δ values (to mitigate outliers).

Both Mean and Median are reported in the tables; effect size d (for the regularization experiments) and p-values are provided separately in the appendix tables.

7.1 Vanilla MLP Experiments (No Regularization)

Table 4: Custom vs. Adam-family Δ% (Dropout & Label Smoothing OFF) — Mean / Median

Hidden	Baseline	Train	Val	Acc	F_1	Time
7	Adam Mean	−37.8%	+0.0%	+1.2%	+1.4%	−44.6%
	Adam Median	−49.6%	+7.5%	−0.1%	+0.2%	−46.5%
	AdamW Mean	−39.6%	+0.5%	+1.0%	+1.1%	−46.6%
	AdamW Median	−48.4%	+5.2%	−0.2%	−0.3%	−48.1%
	AdaBelief Mean	−39.8%	+4.0%	+0.9%	+1.1%	−47.6%
	AdaBelief Median	−49.1%	+10.6%	+0.1%	−0.0%	−49.2%
10	Adam Mean	−52.3%	+25.3%	+0.6%	+0.7%	−39.7%
	Adam Median	−89.1%	+17.8%	+0.1%	+0.2%	−41.0%
	AdamW Mean	−44.8%	+26.6%	+0.9%	+0.7%	−42.2%
	AdamW Median	−75.5%	+23.1%	+0.6%	+0.4%	−45.9%
	AdaBelief Mean	−56.0%	+30.2%	+0.2%	+0.2%	−44.3%
	AdaBelief Median	−94.9%	+25.9%	−0.1%	−0.1%	−47.2%

Overfitting Characteristics

- **Training Loss**: Median improvement up to −95%, indicating very low training loss.
- **Validation Loss**: For the 10-layer MLP with AdaBelief, Median $\Delta_{\text{Val}} = +25.9\%$ (Mean $\Delta_{\text{Val}} = +30.2\%$), indicating substantial degradation and clear overfitting by the custom optimizer across baselines.
- **Accuracy & F_1**: Median deviation within ±0.3%.
- **Training Time**: Both Mean and Median show a 40–50% reduction.

Thus, overfitting is pronounced, suggesting that regularization is essential.

7.2 Regularization Experiments (Custom: Dropout 0.004 & LS 0.025; Baseline: BatchNorm + Dropout + LS)

Table 5: Custom vs. Baseline Δ% after Regularization — Mean / Median

Hidden	Baseline	Train	Val	Acc	F_1	Time
7	Adam Mean	−58.4%	−19.7%	+0.1%	−0.2%	−57.8%
	Adam Median	−74.8%	−20.6%	−0.2%	−0.2%	−60.2%
	AdamW Mean	−58.4%	−19.3%	+0.3%	−0.2%	−58.9%
	AdamW Median	−74.6%	−20.1%	−0.4%	−0.5%	−60.9%
	AdaBelief Mean	−58.5%	−20.1%	−0.0%	−0.0%	−60.1%
	AdaBelief Median	−75.4%	−20.0%	−0.1%	+0.0%	−61.9%
10	Adam Mean	−54.2%	−19.6%	−1.9%	−1.7%	−64.0%
	Adam Median	−71.9%	−21.0%	−0.8%	−0.4%	−66.3%
	AdamW Mean	−54.2%	−19.8%	−1.7%	−1.7%	−65.8%
	AdamW Median	−71.8%	−21.5%	−0.0%	−0.0%	−68.3%
	AdaBelief Mean	−54.3%	−19.4%	−1.8%	−1.6%	−67.0%
	AdaBelief Median	−71.9%	−21.2%	−0.5%	−0.3%	−69.4%

Generalization and Speed Improvements (Regularization) After applying Dropout and Label Smoothing, the custom optimizer achieved the following Median/Mean improvements per hidden size:

1) **Validation Loss Reduction**

   Range: Median: 20.0%–21.2%; Mean: 19.3%–20.1%

   ⇒ Overall generalization advantage over baselines.

2) **Training Time Reduction**

   Average Reduction: Median = 64.5%, Mean = 62.3%

   ⇒ Up to 66% faster than baselines with BatchNorm.

3) **Stability of Acc. & F_1** Median deviation $\leq \pm 0.5\%$, Mean deviation $\leq \pm 1.9\%$, indicating no accuracy penalty.

7.2.1 Cases Where Custom Was Inferior

Table 6: Datasets where Custom underperformed: relative differences (Δ%), statistical significance (✓), and effect size (★: $|d| \geq 1.2$, ☆: $0.8 \leq |d| < 1.2$)

Dataset	Baseline	ΔVal	ΔAcc	ΔF_1	p_{ttest}	d
Hidden 7						
Gauss_sep1.0_clust3_flip0.05	Adam	+10.0%	−3.0%	−3.0%	4.99×10^{-5} ✓	+2.60★
	AdamW	+9.1%	−3.2%	−3.2%	8.47×10^{-5} ✓	+2.52★
	AdaBelief	+9.3%	−3.0%	−3.0%	1.36×10^{-4} ✓	+2.58★
20NG	Adam	+8.7%	−8.5%	−8.2%	2.21×10^{-3} ✓	+2.77★
	AdamW	+7.7%	−8.1%	−7.7%	7.31×10^{-4} ✓	+2.29★
	AdaBelief	+8.3%	−7.8%	−7.4%	5.63×10^{-4} ✓	+2.64★
Gauss_sep1.0_clust3	Adam	+4.5%	−3.0%	−3.0%	3.47×10^{-2} ✓	+0.85☆
	AdamW	+4.6%	−3.0%	−3.0%	5.49×10^{-2}	+0.84
	AdaBelief	+5.6%	−3.1%	−3.1%	1.42×10^{-2} ✓	+0.97☆
WineQuality	Adam	+1.9%	−1.8%	−2.8%	2.75×10^{-1}	+0.28
Hidden 10						
20NG	Adam	+12.7%	−14.6%	−14.1%	3.52×10^{-6} ✓	+5.65★
	AdamW	+12.0%	−14.5%	−14.1%	2.91×10^{-5} ✓	+5.18★
	AdaBelief	+12.9%	−13.8%	−13.2%	3.81×10^{-5} ✓	+4.98★
Gauss_sep1.0_clust3_flip0.05	Adam	+3.9%	−2.4%	−2.4%	5.35×10^{-3} ✓	+1.12★
	AdamW	+3.7%	−2.4%	−2.4%	8.64×10^{-3} ✓	+1.03★
	AdaBelief	+3.5%	−2.7%	−2.7%	5.06×10^{-3} ✓	+0.94☆

Legend: Δ denotes the percentage relative difference of Custom vs. Baseline; a positive value indicates Custom underperforms (higher Val-Loss or lower accuracy). ✓ indicates statistical significance at $p < 0.05$. ★ ($|d| \geq 1.2$) denotes a very large effect size; ☆ ($0.8 \leq |d| < 1.2$) denotes a large effect size.

Detailed Analysis

1. **Underperforming Cases**: Custom underperformance (Val Loss ↑) is concentrated on 20NG and low-separation Gaussians (including flip=0.05). No underperformance was observed on image datasets (CIFAR-10/100, MNIST).

2. **Significance & Effect Size**: For 20NG and Gauss_sep1.0_clust3_flip0.05, most comparisons yield $p < 10^{-2}$ & $|d| \geq 1.2$ (★). WineQuality shows $p = 0.28$, $|d| = 0.28$, indicating no significance and small effect.

3. **Concurrent Accuracy & F_1 Drops**: Underperforming datasets also exhibit 2–14 %p drops in Acc/F_1, confirming that loss increases translate to real performance degradation.

4. **Overall Advantage Retained**: Excluding the two underperformers, Custom reduces Val-Loss by over 20 % on 10/12 datasets (Table 5).

Detailed numbers and per-epoch loss curves can be found in Appendix E (Sections E.1–E.4).

7.3 Measured Training Time and GPU Utilization

Although the theoretical complexity of our optimizer is $O(B\,r\,n^3)$, in practice we fuse small n^2 operations into a single large tensor contraction on the GPU, minimizing kernel launches. This yields consistent speed advantages over Adam ($O(B\,r\,n^2)$).

Moreover, measured GPU utilization for baselines was around 36% on average, whereas our custom optimizer achieved over 55%, demonstrating significantly higher compute intensity.

7.4 Final Hyperparameter Selection from Sensitivity Analysis

For the four optimizers (Adam, AdamW, AdaBelief, and Custom), analysis of the Dropout–Label Smoothing heatmaps presented in Appendix E.5 revealed that, among the combinations $\{0.002, 0.004, 0.008\} \times \{0.0125, 0.025, 0.05\}$, the pair (0.004, 0.025) exhibited the most **average validation loss performance** across all four optimizers without excessive variance. Therefore, we conservatively adopt Dropout = 0.004 and Label Smoothing $\alpha = 0.025$ as the final hyperparameters.

8 Conclusion and Future Work

8.1 Conclusion Summary

The lightweight, diagonal-only second-order custom optimizer proposed in this work demonstrated the following key results:

- **Vanilla MLP (no regularization)**: Training loss was reduced by up to 95%, but validation loss increased on many datasets, indicating pronounced overfitting (Table 4).
- **With Dropout & Label Smoothing**: Validation loss was decreased by an average of 19.3%–20.1%, and training time was reduced by approximately 62% (mean/median 60%), achieving superior generalization and speed compared to baselines (Table 5).
- **BatchNorm exclusion rationale**: Applying BatchNorm to the custom optimizer caused loss instability and explosion; therefore, we did not include BatchNorm in the custom setting, while baselines used it to ensure a fair comparison.
- **Early stopping effect**: Appendix E.2 and E.4 show that, even on datasets where the custom optimizer underperformed, applying early stopping yields better validation loss than baselines.
- **FP16 TensorCore pilot test**: On the same hardware/software setup using a 7-layer Leaky-ReLU MLP (7 hidden layers $\times$ 256 units, input dim = 784, batch size = 64), a pilot with float16 arithmetic and TensorCore reduced custom optimizer training time to 201s versus 286s for Adam (30% speedup).
- **Medium–large scale scalability**: Given sufficient computational resources, we preliminarily confirmed that, analogous to the 7-layer Leaky-ReLU MLP pilot test, the custom optimizer proposed in this study can be expected to yield consistent training-time reductions relative to the baseline even on medium- and large-scale models.

8.2 Limitations

- Superior performance on some datasets depends on early stopping.
- FP16/TensorCore results are based on a limited pilot; systematic large-scale validation is required.

8.3 Future Work

- **Comparison with other lightweight second-order methods**: Experimental benchmarking against Shampoo, K-FAC, AdaHessian, etc., will be conducted in future work.
- **Low-separation and noisy environment experiments**: Conduct deeper evaluation of generalization on 20 Newsgroups text and low-separation Gaussian datasets.
- **Extension to CNNs and Transformers**: Apply the proposed method to large-scale models (ResNet, Vision Transformer) in vision and language to assess performance, training speed, and memory efficiency.
- **FP16/TensorCore kernel optimization**: Implement custom kernels that exploit float16 and TensorCore to maximize training speed and energy efficiency.
- **Medium–large scale scaling**: Perform comprehensive evaluation and improvement of training time and memory efficiency on various real-world datasets and large MLP/CNN/Transformer architectures.

References

[1] Yann N 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. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2933–2941, 2014.

[2] Behrooz Fazlyab, Carlos Okuno, Yaman Yoshida, Barnabás Póczos, and Milad Razaviyayn. Provable bounds on the hessian of neural networks. *arXiv preprint arXiv:2406.04476*, 2024.

[3] Behnam Ghorbani, Shiva Prasad Krishnan, and Yingyu Xiao. An investigation into neural net hessian spectra in modern architectures. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 2234–2243, 2019.

[4] Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 2956–2964, 2018.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[6] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019. OpenReview: `https://openreview.net/forum?id=Bkg6RiCqY7`.

[7] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 2408–2417, 2015.

[8] Levent Sagun, Léon Bottou, and Yann LeCun. Eigenvalues of the hessian in deep learning: Singularity and beyond. *arXiv preprint arXiv:1611.07476*, 2016.

[9] James Vo. Efficient second-order neural network optimization via adaptive trust region methods. *arXiv preprint arXiv:2410.02293*, 2024.

[10] Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. Adahessian: An adaptive second order optimizer for machine learning. In *International Conference on Learning Representations (ICLR)*, 2021.

[11] Yanjun Zhao, Sizhe Dang, Haishan Ye, Guang Dai, Yi Qian, and Ivor Tsang. Second-order fine-tuning without pain for llms: A hessian informed zeroth-order optimizer. In *International Conference on Learning Representations (ICLR)*, 2025. Poster.

[12] Juntang Zhuang, Tianyun Tang, Yifan Ding, Sekhar Tatikonda, Nicha C. Dvornek, Xenophon Papademetris, and James S. Duncan. Adabelief optimizer: Adapting step-sizes by the belief in observed gradients. *arXiv preprint arXiv:2010.07468*, 2020.

A Proof of Global Convergence of Newton's Method

1. Problem Setup and Assumptions

Assume the function $f\colon \mathbb{R}^n \to \mathbb{R}$ satisfies the following conditions.

1. **Strong Convexity**: There exists a constant $m > 0$ such that

$$\nabla^2 f(x) \succeq mI, \quad \forall x \in \mathbb{R}^n.$$

   That is, f is m-strongly convex, so the minimizer x^* is unique and

$$\nabla f(x^*) = 0.$$

2. **Twice Continuously Differentiable**: $f \in C^2(\mathbb{R}^n)$.
3. **Hessian Upper Bound**: There exists a constant $L > 0$ such that

$$\nabla^2 f(x) \preceq L\, I, \quad \forall x \in \mathbb{R}^n.$$

Instead of the standard Newton update

$$x_{t+1} = x_t - \alpha \left[\nabla^2 f(x_t)\right]^{-1} \nabla f(x_t),$$

we consider an update using only the diagonal entries of the Hessian.

2. Diagonal Update Algorithm

2.1 Definition of the Diagonal Approximation Matrix D_t

At each iteration t, collect the diagonal elements of $\nabla^2 f(x_t)$ into a diagonal matrix $D_t \in \mathbb{R}^{n\times n}$:

$$D_t := \operatorname{diag}\big(\tfrac{\partial^2 f}{\partial x_1^2}(x_t),\ \tfrac{\partial^2 f}{\partial x_2^2}(x_t),\ \ldots,\ \tfrac{\partial^2 f}{\partial x_n^2}(x_t)\big).$$

From strong convexity $\nabla^2 f(x) \succeq mI$ and the Hessian upper bound $\nabla^2 f(x) \preceq L\, I$, it follows for all t that

$$m\, e_i^T e_i = m \le e_i^T \nabla^2 f(x_t)\, e_i = \tfrac{\partial^2 f}{\partial x_i^2}(x_t) \le L\, e_i^T e_i = L, \quad i = 1, \ldots, n,$$

where e_i are the standard basis vectors. Therefore,

$$m\, I \preceq D_t \preceq L\, I, \quad \lambda_{\min}(D_t) = \min_{1\le i\le n} D_{t,ii} \ge m, \quad \lambda_{\max}(D_t) = \max_{1\le i\le n} D_{t,ii} \le L.$$

2.2 Diagonal Approximation Update Rule

With a fixed scalar step size $\alpha > 0$, update as follows:

$$x_{t+1} = x_t - \alpha\, D_t^{-1}\, \nabla f(x_t).$$

Define the error by

$$e_t := \left\|x_t - x^*\right\|.$$

We prove that this update exhibits global linear convergence under an appropriate choice of α.

3. Proof

Since $f \in C^2$ and $\nabla^2 f(x) \preceq L\, I$ (hence f is L-smooth), for any $x \in \mathbb{R}^n$ and $\Delta \in \mathbb{R}^n$, by the multivariate Lagrange remainder (see Appendix B) we have

$$f(x+\Delta) = f(x) + \nabla f(x)^T \Delta + \tfrac{1}{2}\, \Delta^T \nabla^2 f(c)\, \Delta, \quad c \in \{x + \theta\Delta : 0 < \theta < 1\}.$$

Hence

$$\Delta^T \nabla^2 f(c)\, \Delta \le L\, \|\Delta\|^2 \quad \Longrightarrow \quad f(x+\Delta) \le f(x) + \nabla f(x)^T \Delta + \tfrac{L}{2}\, \|\Delta\|^2.$$

Substitute $x = x_t$, $\Delta = -\alpha\, D_t^{-1}\, \nabla f(x_t)$:

$$\begin{aligned} f(x_{t+1}) &= f\big(x_t - \alpha\, D_t^{-1}\, \nabla f(x_t)\big) \\ &\le f(x_t) + \nabla f(x_t)^T \big(-\alpha\, D_t^{-1}\, \nabla f(x_t)\big) + \tfrac{L}{2}\, \big\|-\alpha\, D_t^{-1}\, \nabla f(x_t)\big\|^2 \\ &= f(x_t) - \alpha\, \nabla f(x_t)^T\, D_t^{-1}\, \nabla f(x_t) + \tfrac{L\alpha^2}{2}\, \big\|D_t^{-1}\, \nabla f(x_t)\big\|^2. \end{aligned}$$

Since $m\, I \preceq D_t \preceq L\, I$, we have

$$\|D_t^{-1}\| \le \tfrac{1}{m}, \quad \text{(where } \|\cdot\| \text{ denotes the spectral norm)},$$

$$\nabla f(x_t)^T D_t^{-1} \nabla f(x_t) \ge \tfrac{1}{\lambda_{\max}(D_t)}\, \|\nabla f(x_t)\|^2 \ge \tfrac{1}{L}\, \|\nabla f(x_t)\|^2.$$

Moreover,

$$\big\|D_t^{-1}\, \nabla f(x_t)\big\|^2 \le \|D_t^{-1}\|^2\, \|\nabla f(x_t)\|^2 \le \tfrac{1}{m^2}\, \|\nabla f(x_t)\|^2.$$

Thus

$$\begin{aligned} f(x_{t+1}) &\le f(x_t) - \alpha\, \nabla f(x_t)^T D_t^{-1}\, \nabla f(x_t) + \tfrac{L\alpha^2}{2}\, \|D_t^{-1}\, \nabla f(x_t)\|^2 \\ &\le f(x_t) - \alpha\, \tfrac{1}{L}\, \|\nabla f(x_t)\|^2 + \tfrac{L\alpha^2}{2}\, \tfrac{1}{m^2}\, \|\nabla f(x_t)\|^2 \\ &= f(x_t) - \Big(\tfrac{\alpha}{L} - \tfrac{L\alpha^2}{2m^2}\Big)\, \|\nabla f(x_t)\|^2. \end{aligned}$$

Similarly, by the multivariate Lagrange remainder,

$$f(x_t) = f(x^*) + \nabla f(x^*)^T (x_t - x^*) + \tfrac{1}{2}\, (x_t - x^*)^T \nabla^2 f(c')\, (x_t - x^*), \quad c' \in \{x^* + \theta'(x_t - x^*) : 0 < \theta' < 1\}.$$

Since $\nabla f(x^*) = 0$, it follows that

$$f(x_t) - f(x^*) = \tfrac{1}{2}\, (x_t - x^*)^T \nabla^2 f(c')\, (x_t - x^*).$$

Hence

$$\tfrac{m}{2}\, \|x_t - x^*\|^2 \le f(x_t) - f(x^*) \le \tfrac{L}{2}\, \|x_t - x^*\|^2.$$

Moreover,

$$\|\nabla f(x_t)\|^2 \ge \tfrac{m}{2}\, \big(f(x_t) - f(x^*)\big),$$

so

$$\begin{aligned} f(x_{t+1}) - f(x^*) &\le f(x_t) - f(x^*) - \Big(\tfrac{\alpha}{L} - \tfrac{L\alpha^2}{2m^2}\Big)\, \tfrac{m}{2}\, \big(f(x_t) - f(x^*)\big) \\ &= \Big[1 - \Big(\tfrac{\alpha}{L} - \tfrac{L\alpha^2}{2m^2}\Big)\, \tfrac{m}{2}\Big]\, \big(f(x_t) - f(x^*)\big). \end{aligned}$$

Let $r(\alpha) := \left(\frac{\alpha}{L} - \frac{L\alpha^2}{2m^2}\right)\frac{m}{2}$. Then

$$f(x_{t+1}) - f(x^*) \le \big[1 - r(\alpha)\big]\big(f(x_t) - f(x^*)\big).$$

For $r(\alpha) > 0$, we require

$$0 < \alpha < \tfrac{2m^2}{L^2}.$$

In particular, for $\alpha = \frac{m^2}{L^2}$,

$$r(\alpha) = \left(\frac{m^2/L^2}{L} - \frac{L\,(m^2/L^2)^2}{2m^2}\right)\frac{m}{2} = \frac{m^3}{4L^3},$$

Hence, $r(\alpha)$ attains its maximum, and

$$0 < 1 - \tfrac{m^3}{4\,L^3} < 1.$$

Therefore,

$$\rho := 1 - r(\alpha), \quad 0 < \rho < 1, \quad f(x_{t+1}) - f(x^*) \;\le\; \rho\big(f(x_t) - f(x^*)\big),$$

i.e., the function value difference converges linearly.

Moreover,

$$\tfrac{m}{2}\,\|x_t - x^*\|^2 \le f(x_t) - f(x^*) = O(\rho^t) \implies \|x_t - x^*\| \to 0 \quad (t \to \infty).$$

Thus x_t converges globally at a linear rate.

The admissible range of α is

$$0 < \alpha < \tfrac{2m^2}{L^2}.$$

Appendix: Proof of $\|\nabla f(x_t)\|^2 \ge \frac{m}{2}\big[f(x_t) - f(x^*)\big]$

Let $g(t) = f\big(x^* + t(x_t - x^*)\big)$.

$$g''(t) = (x_t - x^*)^T \nabla^2 f\big(x^* + t(x_t - x^*)\big)\,(x_t - x^*) \;\ge\; m\,\|x_t - x^*\|^2 \;\ge\; 0, \quad \text{(hence } g' \text{ is increasing).}$$

$$g(1) - g(0) = f(x_t) - f(x^*) = g'(c) = \nabla f\big(x^* + c(x_t - x^*)\big)^T (x_t - x^*)$$

$$\le g'(1) = \nabla f(x_t)^T (x_t - x^*) \le \|\nabla f(x_t)\|\,\|x_t - x^*\|.$$

$$\therefore \quad \tfrac{m}{2}\,\|x_t - x^*\|^2 \;\le\; f(x_t) - f(x^*), \tfrac{m}{2}\left[f(x_t) - f(x^*)\right] \;\le\; \tfrac{m}{2}\,\|\nabla f(x_t)\|\,\|x_t - x^*\|,$$

$$\tfrac{m}{2}\,\|x_t - x^*\|\,\|\nabla f(x_t)\| \;\le\; \|\nabla f(x_t)\|^2, \quad \therefore\; \tfrac{m}{2}\left[f(x_t) - f(x^*)\right] \;\le\; \|\nabla f(x_t)\|^2.$$

which completes the proof.

B Multivariate Lagrange Remainder Theorem

Multivariate Taylor Expansion Theorem (Lagrange Remainder Form)

Theorem. Let $f\colon D \to \mathbb{R}$ be a C^{n+1} function on $D \subset \mathbb{R}^d$, and assume that for any $a, x \in D$,

$$\big\{\, a + t(x - a) \;\big|\; t \in [0,1] \big\} \;\subset D.$$

Then there exists some $\theta \in (0,1)$ such that

$$f(x) \;=\; \sum_{|\alpha|\le n} \frac{D^\alpha f(a)}{\alpha!}\,(x-a)^\alpha \;+\; \sum_{|\alpha|=n+1} \frac{D^\alpha f\big(a+\theta(x-a)\big)}{\alpha!}\,(x-a)^\alpha, \tag{6}$$

where $\alpha = (\alpha_1, \ldots, \alpha_d) \in \mathbb{N}^d$ is a multi–index with

$$|\alpha| = \alpha_1 + \cdots + \alpha_d, \quad \alpha! = \alpha_1! \cdots \alpha_d!, \quad (x-a)^\alpha = (x_1-a_1)^{\alpha_1} \cdots (x_d-a_d)^{\alpha_d},$$

and $D^\alpha f = \frac{\partial^{|\alpha|} f}{\partial x_1^{\alpha_1} \cdots \partial x_d^{\alpha_d}}$.

Proof.

(i) **Derivative calculation for the univariate function.**

First, define the univariate function g as

$$g\colon [0,1] \to \mathbb{R}, \qquad g(t) = f\big(a + t(x-a)\big).$$

Since $f \in C^{n+1}(D)$ and the line segment $\{\, a + t(x-a) \mid t \in [0,1]\} \subset D$, it follows that $g \in C^{n+1}[0,1]$.

By the chain rule and multi–index notation, for any $k = 0, \ldots, n+1$,

$$g^{(k)}(t) = \sum_{|\alpha|=k} \frac{k!}{\alpha!}\, D^\alpha f\big(a + t(x-a)\big)\,(x-a)^\alpha.$$

In particular, at $t = 0$,

$$\frac{g^{(k)}(0)}{k!} = \sum_{|\alpha|=k} \frac{D^\alpha f(a)}{\alpha!}\,(x-a)^\alpha.$$

(ii) **Univariate Taylor theorem (Lagrange remainder form).**

Since $g \in C^{n+1}[0,1]$, applying the univariate Taylor theorem with Lagrange remainder (see Appendix C) yields the existence of $\theta \in (0,1)$ such that

$$g(1) = \sum_{k=0}^{n} \frac{g^{(k)}(0)}{k!} + \frac{g^{(n+1)}(\theta)}{(n+1)!}\,(1-0)^{n+1}.$$

(iii) **Rewriting in the multivariate form.**

Noting $g(1) = f(x)$ and substituting the expressions for $\frac{g^{(k)}(0)}{k!}$ and $g^{(n+1)}(\theta)$ from (i) gives

$$f(x) = \sum_{k=0}^{n} \sum_{|\alpha|=k} \frac{D^\alpha f(a)}{\alpha!}\,(x-a)^\alpha + \sum_{|\alpha|=n+1} \frac{D^\alpha f\big(a+\theta(x-a)\big)}{\alpha!}\,(x-a)^\alpha.$$

This is exactly (6). □

C Univariate Lagrange Remainder Theorem

Univariate Taylor's Theorem (Lagrange Remainder Form)

Taylor's Theorem (Lagrange remainder). Let $f\colon [a,b] \to \mathbb{R}$ be $(n+1)$-times continuously differentiable. Then for any $x \in [a,b]$ one has

$$f(x) \;=\; f(a) \;+\; f'(a)\,(x-a) \;+\; \frac{f''(a)}{2!}(x-a)^2 \;+\; \cdots \;+\; \frac{f^{(n)}(a)}{n!}(x-a)^n \;+\; R_{n+1}(x),$$

where the remainder $R_{n+1}(x)$ is given by

$$R_{n+1}(x) \;=\; \frac{f^{(n+1)}(c)}{(n+1)!}\,(x-a)^{n+1}, \quad \text{for some } c \in (a,x) \text{ (or } c \in (x,a)).$$

In particular, one may write $c = a + \theta(x-a)$ with $0 < \theta < 1$.

Proof. The proof proceeds by mathematical induction.

※ Note on the boundary point $x = a$.
If $x = a$, then every factor $(x-a)$ on the right–hand side vanishes, so the remainder $R_{n+1}(a)$ is trivially 0. Thus at $x = a$ we already have

$$f(a) = f(a) + 0 + \cdots + 0 + R_{n+1}(a), \quad R_{n+1}(a) = 0,$$

and it suffices to concentrate the induction argument on the case $x \neq a$.

1. Base Case ($n = 0$).
Assume f is once continuously differentiable. For $n = 0$,

$$f(x) = f(a) \;+\; R_1(x), \quad R_1(x) = f(x) - f(a).$$

By the Mean Value Theorem,

$$\frac{f(x) - f(a)}{x - a} = f'(\xi), \quad \xi \in (a,x),$$

hence

$$R_1(x) = f(x) - f(a) = f'(\xi)\,(x-a) = \frac{f^{(1)}(\xi)}{1!}\,(x-a), \quad \xi \in (a,x),$$

showing the result for $n = 0$.

2. Induction Hypothesis ($n = k-1$).
Suppose for some integer $k \geq 1$ that f is k-times continuously differentiable and

$$f(x) = \sum_{i=0}^{k-1} \frac{f^{(i)}(a)}{i!}\,(x-a)^i \;+\; R_k(x), \quad R_k(x) = \frac{f^{(k)}(c)}{k!}\,(x-a)^k, \quad c \in (a,x).$$

3. Induction Step ($n = k$).
Assume f is $(k+1)$-times continuously differentiable. Recall the induction hypothesis:

$$f(x) = \sum_{i=0}^{k-1} \frac{f^{(i)}(a)}{i!}\,(x-a)^i \;+\; R_k(x), \quad R_k(x) = \frac{f^{(k)}(c)}{k!}\,(x-a)^k, \quad c \in (a,x).$$

We wish to split $R_k(x)$ into the $i = k$ Taylor term plus a new $(k+1)$th remainder:

$$f(x) = \sum_{i=0}^{k} \frac{f^{(i)}(a)}{i!}\,(x-a)^i \;+\; R_{k+1}(x), \quad R_{k+1}(x) = \frac{f^{(k+1)}(c')}{(k+1)!}\,(x-a)^{k+1}, \quad c' \in (a,x).$$

This is done in two steps.

(Step 3-1) Apply the Mean Value Theorem to $R_k(x)$.

$$R_k(x) = \frac{f^{(k)}(c)}{k!}\,(x-a)^k, \quad c \in (a,x).$$

Since $f^{(k)}$ is continuously differentiable on $a < c < x$, the Mean Value Theorem yields

$$\frac{f^{(k)}(c) - f^{(k)}(a)}{c-a} = f^{(k+1)}(\xi), \quad \xi \in (a,c) \subset (a,x).$$

※ Note on the case $x < a$.

> Throughout this proof, each instance of "$\xi \in (a,x)$" should be read as "$\xi \in (x,a)$" when $x < a$. Only the direction of the interval changes; the argument remains the same. One may uniformly write "$\xi \in (\min\{a,x\},\, \max\{a,x\})$."

Hence

$$f^{(k)}(c) = f^{(k)}(a) + f^{(k+1)}(\xi)\,(c-a).$$

Since $c \in (a,x)$, one may write $c = a + \theta\,(x-a)$ $(0 < \theta < 1)$. Then $c - a = \theta\,(x-a)$, and hence

$$f^{(k)}(c) = f^{(k)}(a) + f^{(k+1)}(\xi)\,\theta\,(x-a), \quad \xi \in (a,c) \subset (a,x).$$

Substituting into $R_k(x)$ gives

$$\begin{aligned} R_k(x) &= \frac{f^{(k)}(c)}{k!}\,(x-a)^k = \frac{1}{k!}\Big[f^{(k)}(a) + f^{(k+1)}(\xi)\,\theta\,(x-a)\Big]\,(x-a)^k \\ &= \underbrace{\frac{f^{(k)}(a)}{k!}\,(x-a)^k}_{\text{Taylor term } i=k} \;+\; \underbrace{\frac{f^{(k+1)}(\xi)}{k!}\,\theta\,(x-a)^{k+1}}_{\text{new } (k+1)\text{th remainder}}. \end{aligned}$$

The first term

$$\frac{f^{(k)}(a)}{k!}(x-a)^k$$

is already the $i = k$ term of $\displaystyle\sum_{i=0}^{k} \frac{f^{(i)}(a)}{i!}(x-a)^i$, so it can be absorbed. Hence the actual $(k+1)$th remainder is

$$R_{k+1}(x) = \frac{f^{(k+1)}(\xi)}{k!}\,\theta\,(x-a)^{k+1}, \quad \xi \in (a,x),\ \theta \in (0,1).$$

This is the form "obtained solely via the Mean Value Theorem," with $\xi = a + \theta(x-a)$.

※ Additional explanation regarding the roles of c and ξ.

- In Step 3-1,
$$R_k(x) = \frac{f^{(k)}(c)}{k!}\,(x-a)^k, \quad c \in (a,x)$$
the point c is already known by the induction hypothesis to lie in (a,x). In the subsequent application of the Mean Value Theorem, two new quantities appear:
$$c = a + \theta\,(x-a), \quad 0 < \theta < 1, \quad \xi \in (a,c) \subset (a,x).$$

- To match the "Lagrange remainder form," one then combines θ and ξ into a single new point c'. From Step 3-2 we have shown $\theta = \frac{1}{k+1}$. Hence one may take
$$c' = a + \frac{1}{k+1}(x-a) \in (a,x),$$
or apply the Mean Value Theorem once more to identify an actual $c' \in (a,x)$, yielding
$$R_{k+1}(x) = \frac{f^{(k+1)}(c')}{(k+1)!}\,(x-a)^{k+1}.$$

(Step 3-2) Determination of $\theta = \frac{1}{k+1}$ from the $(k+1)$th-derivative consistency.

※ Additional explanation of the required existence of higher derivatives.
When expanding
$$\frac{d^{k+1}}{dx^{k+1}}\big[f^{(k+1)}(a+\theta(x-a))\,(x-a)^{k+1}\big]$$
via Leibniz's rule, it may appear that one needs derivatives of f up to order $k+1+j$ (i.e., up to order $2k+2$) for the term $G^{(j)}(x)$. However, since the only evaluation point is $x = a$, we have
$$\Psi(x) = (x-a)^{k+1} \quad \Longrightarrow \quad \Psi^{(m)}(a) = 0 \quad (0 \le m \le k),$$
so all terms with $j \ge 1$ include at least one factor of $(x-a)$ and thus vanish at $x = a$. Consequently, the existence of $G^{(j)}(a)$ requires only the continuity of $f^{(k+1)}$. Therefore, the assumption that f is $(k+1)$-times continuously differentiable suffices for this step.

Although we already know that
$$R_{k+1}(x) = \frac{f^{(k+1)}(\xi)}{k!}\,\theta\,(x-a)^{k+1},$$
one may also assume from the outset
$$R_{k+1}(x) = \theta\,\frac{f^{(k+1)}\big(a+\theta(x-a)\big)}{k!}\,(x-a)^{k+1}, \quad 0 < \theta < 1,$$
and then differentiate both sides $(k+1)$ times to show explicitly that $\theta = \frac{1}{k+1}$. That is, if the above holds then
$$\frac{d^{k+1}}{dx^{k+1}}[\,f(x)\,] \;=\; \frac{d^{k+1}}{dx^{k+1}}\Big[\sum_{i=0}^{k}\frac{f^{(i)}(a)}{i!}(x-a)^i \;+\; \theta\,\frac{f^{(k+1)}(a+\theta(x-a))}{k!}\,(x-a)^{k+1}\Big].$$

The left–hand side is

$$\frac{d^{k+1}}{dx^{k+1}}\big[f(x)\big] = f^{(k+1)}(x).$$

On the right, the polynomial $\sum_{i=0}^{k} \frac{f^{(i)}(a)}{i!}(x-a)^i$ vanishes after $(k+1)$ derivatives, leaving

$$\theta\,\frac{d^{k+1}}{dx^{k+1}}\Big[\frac{1}{k!}\,f^{(k+1)}\big(a+\theta(x-a)\big)\,(x-a)^{k+1}\Big].$$

To compute this, set

$$G(x) := f^{(k+1)}\big(a+\theta(x-a)\big), \qquad \Psi(x) := (x-a)^{k+1}.$$

Then by Leibniz's rule,

$$\frac{d^{k+1}}{dx^{k+1}}\big[G(x)\,\Psi(x)\big] = \sum_{j=0}^{k+1}\binom{k+1}{j}\,G^{(j)}(x)\,\Psi^{(k+1-j)}(x).$$

Since

$$\Psi(x) = (x-a)^{k+1} \quad\Longrightarrow\quad \Psi^{(m)}(x) = \begin{cases} \dfrac{(k+1)!}{(k+1-m)!}\,(x-a)^{k+1-m}, & 0 \le m \le k, \\ (k+1)!, & m = k+1, \end{cases}$$

the only nonzero term at $x = a$ comes from $j = 0$. Thus we split the sum as

$$\underbrace{\binom{k+1}{0}\,G^{(0)}(x)\,\Psi^{(k+1)}(x)}_{j=0\text{ term}} + \underbrace{\sum_{j=1}^{k+1}\binom{k+1}{j}\,G^{(j)}(x)\,\Psi^{(k+1-j)}(x)}_{=:g(x)}.$$

- $j = 0$ term: $\binom{k+1}{0} = 1$, $G^{(0)}(x) = G(x)$, $\Psi^{(k+1)}(x) = (k+1)!$, hence

$$j = 0 \text{ term} = (k+1)!\,f^{(k+1)}\big(a+\theta(x-a)\big).$$

- $j \ge 1$ terms define

$$g(x) := \sum_{j=1}^{k+1}\binom{k+1}{j}\,G^{(j)}(x)\,\Psi^{(k+1-j)}(x).$$

  For $1 \le j \le k$,

$$\Psi^{(k+1-j)}(x) = \frac{(k+1)!}{j!}\,(x-a)^j,$$

  so at least one factor of $(x-a)$ remains and therefore $\Psi^{(k+1-j)}(a) = 0$. When $j = k+1$, $\Psi^{(k+1-(k+1))}(x) = \Psi^{(0)}(x) = (x-a)^{k+1}$ also vanishes at $x = a$. Consequently, $g(a) = 0$.

Hence

$$\frac{d^{k+1}}{dx^{k+1}}\big[G(x)\,\Psi(x)\big] = (k+1)!\,f^{(k+1)}\big(a+\theta(x-a)\big) + g(x).$$

Substituting back gives

$$f^{(k+1)}(x) = \theta\,\frac{1}{k!}\Big[(k+1)!\,f^{(k+1)}\big(a+\theta(x-a)\big) + g(x)\Big],$$

i.e.

$$f^{(k+1)}(x) = (k+1)\,\theta\;f^{(k+1)}\big(a+\theta(x-a)\big) + \theta\,\frac{1}{k!}\,g(x).$$

This simplifies to

$$\boxed{f^{(k+1)}(x) = (k+1)\,\theta\,f^{(k+1)}\big(a+\theta(x-a)\big) + \theta\,\frac{1}{k!}\,g(x), \quad g(x) := \sum_{j=1}^{k+1}\binom{k+1}{j}\,G^{(j)}(x)\;\Psi^{(k+1-j)}(x).}$$

(Step 3) Evaluation at $x = a$ and determination of θ
Substituting $x = a$ into the above equation,

$$f^{(k+1)}(a) = (k+1)\,\theta\,f^{(k+1)}\big(a+\theta(a-a)\big) + \theta\,\frac{1}{k!}\,g(x) = (k+1)\,\theta\,f^{(k+1)}(a) + 0 \quad (\because g(a) = 0).$$

Hence, for any C^{k+1} function f, the following identity must hold:

$$f^{(k+1)}(a) \;-\; (k+1)\,\theta\,f^{(k+1)}(a) \;=\; 0 \quad\Longleftrightarrow\quad f^{(k+1)}(a)\,\big[\,1-(k+1)\,\theta\,\big] = 0.$$

The key point is that this equation must be true regardless of whether $f^{(k+1)}(a) = 0$ or $f^{(k+1)}(a) \neq 0$. Therefore, the coefficient

$$1-(k+1)\,\theta$$

must be zero, yielding

$$\theta = \frac{1}{k+1}.$$

Thus the final remainder term is

$$\begin{aligned} R_{k+1}(x) &= \theta\,\frac{f^{(k+1)}\big(a+\theta(x-a)\big)}{k!}\,(x-a)^{\,k+1} \\ &= \frac{1}{k+1}\,\frac{f^{(k+1)}\Big(a+\frac{1}{k+1}(x-a)\Big)}{k!}\,(x-a)^{\,k+1} \\ &= \frac{f^{(k+1)}(c)}{(k+1)!}\,(x-a)^{\,k+1}, \quad c = a+\frac{1}{k+1}(x-a) \in (a,x). \end{aligned}$$

Hence

$$f(x) = \sum_{i=0}^{k}\frac{f^{(i)}(a)}{i!}\,(x-a)^{i} \;+\; \frac{f^{(k+1)}(c)}{(k+1)!}\,(x-a)^{\,k+1}, \quad c = a+\frac{1}{k+1}(x-a) \in (a,x),$$

showing that the Taylor–Lagrange theorem holds for $n = k$. By mathematical induction, the theorem is thus established for all integers $n \geq 0$.

$$
\begin{aligned}
f(x) = & \sum_{i=0}^{k} \frac{f^{(i)}(a)}{i!} (x-a)^i \\
& + \frac{f^{(k+1)}(c)}{(k+1)!} (x-a)^{k+1}, \\
& c = a + \frac{1}{k+1}(x-a) \in (a,x).
\end{aligned}
$$

□

D Dataset Sources

Table 7: Dataset Sources (URL, Authors, Year, License)

Dataset	Source Details
MNIST	LeCun et al. (1998), `http://yann.lecun.com/exdb/mnist`, CC BY-SA 3.0
CIFAR-10/100	Krizhevsky et al. (2009), `https://www.cs.toronto.edu/~kriz/cifar.html`, MIT License
20 Newsgroups	Lang (1995), `http://qwone.com/~jason/20Newsgroups/`, Public Domain
Imbalance	scikit-learn example, `https://scikit-learn.org/`, BSD License
WineQuality-Red	Cortez et al. (2009), UCI ML Repo, `https://archive.ics.uci.edu/ml/datasets/Wine+Quality`, Public Domain
Fashion-MNIST	Xiao et al. (2017), `https://github.com/zalandoresearch/fashion-mnist`, MIT License
HAR	Anguita et al. (2013), UCI ML Repo, `https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones`, Public Domain
Synthetic Gaussian	Generated in this work (scikit-learn BSD License)

E Extended Results (250 epochs)

E.1 Hidden7 Without Regularization

TRAIN Results

	Custom	Adam	Adamw	Adabelief	p_c_vs_a	p_c_vs_aw	p_c_vs_ab
MNIST	0.0	0.0	0.0	0.0	0.0	0.0	0.0
CIFAR10	0.7277	0.7849	0.7933	0.7213	0.0373	0.0083	0.7472
CIFAR100	2.45	2.448	2.4446	2.3281	0.9612	0.901	0.0149
20NG	0.0905	0.0937	0.0908	0.0939	0.1326	0.0031	0.2458
Imbalance	0.0	0.0063	0.0055	0.0013	0.3577	0.1806	0.3648
WineQuality	0.0029	0.0045	0.004	0.0052	0.0781	0.2337	0.0928
FashionMNIST	0.0207	0.0411	0.0401	0.0407	0.0028	0.0028	0.0009
HAR	0.0511	0.0184	0.0207	0.0203	0.0004	0.001	0.0002
Gauss_sep0.5_clust1	0.0001	0.0368	0.0249	0.0161	0.0252	0.0239	0.1893
Gauss_sep1.0_clust3	0.0002	0.0581	0.0519	0.0691	0.012	0.0003	0.0012
Gauss_sep2.0_clust5	0.0001	0.0085	0.0088	0.0213	0.1743	0.1017	0.0463
Gauss_sep1.0_clust3_flip0.05	0.0003	0.101	0.1165	0.1292	0.0	0.0027	0.0

VAL Results

	Custom	Adam	Adamw	Adabelief	p_c_vs_a	p_c_vs_aw	p_c_vs_ab
MNIST	0.2003	0.4329	0.428	0.4111	0.0	0.0	0.0
CIFAR10	2.5121	2.327	2.334	2.3341	0.0882	0.0776	0.0403
CIFAR100	3.917	4.2137	4.2053	4.2443	0.0	0.0009	0.0
20NG	5.5281	5.1607	5.3914	4.964	0.1316	0.428	0.0319
Imbalance	0.5022	0.8548	0.7853	0.7864	0.0168	0.0504	0.0112
WineQuality	2.8786	3.1925	3.2007	3.0861	0.1774	0.0616	0.1311
FashionMNIST	1.1088	0.8888	0.9017	0.8121	0.0011	0.0068	0.0005
HAR	0.2507	0.2094	0.2063	0.219	0.0039	0.0124	0.0715
Gauss_sep0.5_clust1	0.9584	0.7298	0.8071	0.8728	0.0002	0.0964	0.2043
Gauss_sep1.0_clust3	2.3376	2.0799	1.8611	1.8493	0.0219	0.0003	0.0
Gauss_sep2.0_clust5	0.7562	0.7503	0.7352	0.5986	0.8987	0.807	0.0004
Gauss_sep1.0_clust3_flip0.05	3.3487	3.1043	3.0213	2.86	0.0029	0.0002	0.0

ACC Results

	Custom	Adam	Adamw	Adabelief	p_c_vs_a	p_c_vs_aw	p_c_vs_ab
MNIST	0.9775	0.9803	0.9801	0.9794	0.0038	0.0029	0.0324
CIFAR10	0.4819	0.4529	0.4518	0.463	0.0002	0.0002	0.0001
CIFAR100	0.1917	0.1821	0.1826	0.1908	0.0094	0.0014	0.7564
20NG	0.5482	0.5515	0.5528	0.5473	0.1866	0.1749	0.8173
Imbalance	0.9525	0.958	0.9537	0.9585	0.0023	0.6124	0.0001
WineQuality	0.6576	0.6571	0.6683	0.6437	0.9775	0.4457	0.2389
FashionMNIST	0.8784	0.8831	0.8836	0.8841	0.0084	0.0049	0.0008
HAR	0.9397	0.9573	0.9564	0.956	0.0003	0.0018	0.0
Gauss_sep0.5_clust1	0.9005	0.8939	0.8942	0.901	0.157	0.2875	0.9212
Gauss_sep1.0_clust3	0.7852	0.7686	0.7711	0.7665	0.0216	0.0034	0.0003
Gauss_sep2.0_clust5	0.9161	0.9216	0.9189	0.9163	0.1705	0.4596	0.9508
Gauss_sep1.0_clust3_flip0.05	0.7158	0.6872	0.6896	0.6869	0.0001	0.0	0.0

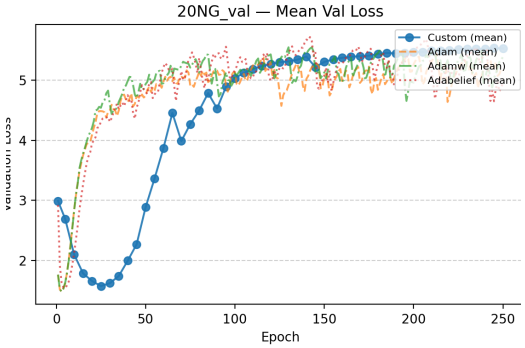
F1 Results

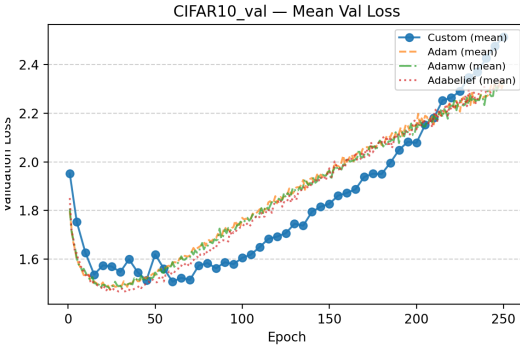
	Custom	Adam	Adamw	Adabelief	p_c_vs_a	p_c_vs_aw	p_c_vs_ab
MNIST	0.9773	0.9801	0.9799	0.9792	0.0039	0.0026	0.0293
CIFAR10	0.4814	0.4514	0.4513	0.4614	0.0	0.0002	0.0001
CIFAR100	0.1839	0.1745	0.176	0.1847	0.0104	0.0064	0.7922
20NG	0.5427	0.5464	0.5481	0.5425	0.1414	0.1035	0.9572
Imbalance	0.9017	0.9139	0.9046	0.9152	0.0013	0.5781	0.0003
WineQuality	0.4352	0.4241	0.437	0.4082	0.4657	0.9263	0.1567
FashionMNIST	0.8783	0.8831	0.8837	0.8842	0.0052	0.004	0.001
HAR	0.9395	0.958	0.9571	0.9569	0.0003	0.0009	0.0
Gauss_sep0.5_clust1	0.9005	0.8938	0.8941	0.9009	0.1547	0.2853	0.935
Gauss_sep1.0_clust3	0.7852	0.7684	0.771	0.7665	0.0213	0.003	0.0004
Gauss_sep2.0_clust5	0.9161	0.9216	0.9189	0.9163	0.1723	0.4578	0.9546
Gauss_sep1.0_clust3_flip0.05	0.7157	0.6868	0.6893	0.6867	0.0001	0.0	0.0

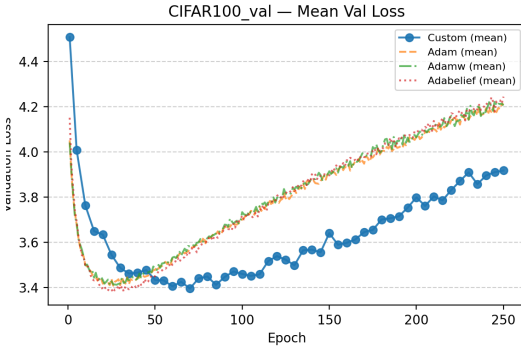
TIME Results

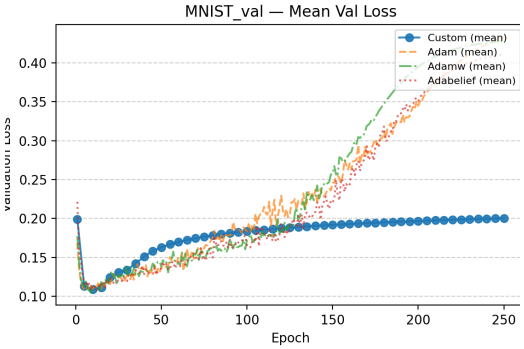
	Custom	Adam	Adamw	Adabelief
MNIST	144.1717	247.7555	254.9061	259.5063
CIFAR10	169.7086	256.3487	261.5442	265.9932
CIFAR100	197.1079	276.4862	287.2436	289.3341
20NG	50.1049	86.0445	91.1562	93.4816
Imbalance	55.3221	112.9044	115.5945	118.6108
WineQuality	7.3027	11.9501	13.1443	14.1618
FashionMNIST	150.2349	270.2277	276.7918	280.9411
HAR	25.5033	49.6695	51.5042	50.9249
Gauss_sep0.5_clust1	54.0626	109.2435	113.3247	114.6174
Gauss_sep1.0_clust3	53.0335	109.8698	113.3813	114.7416
Gauss_sep2.0_clust5	52.844	110.2151	112.9742	114.4973
Gauss_sep1.0_clust3_flip0.05	52.8154	109.7917	112.6658	113.6371

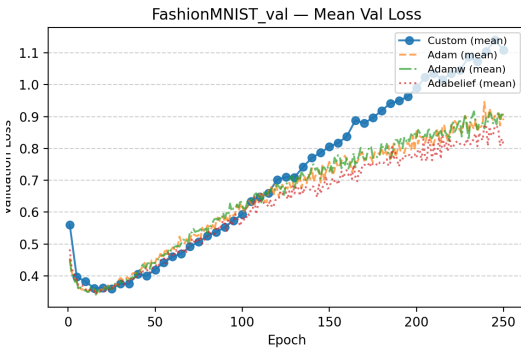
Figure 1: Summary of Experimental Results for Hidden7 Without Regularization

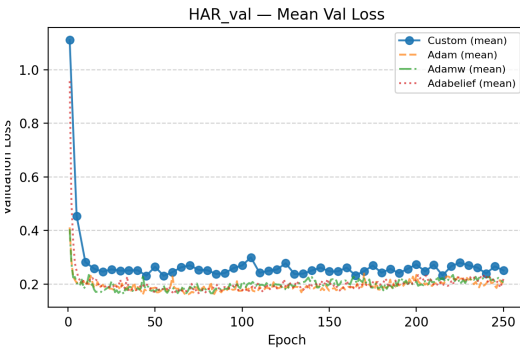

20NG_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

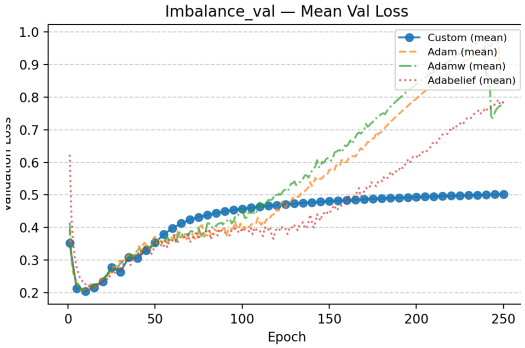

CIFAR10_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

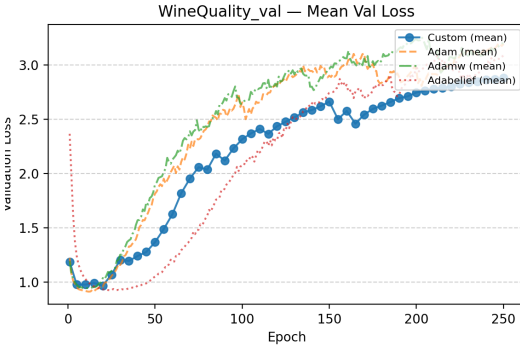

CIFAR100_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

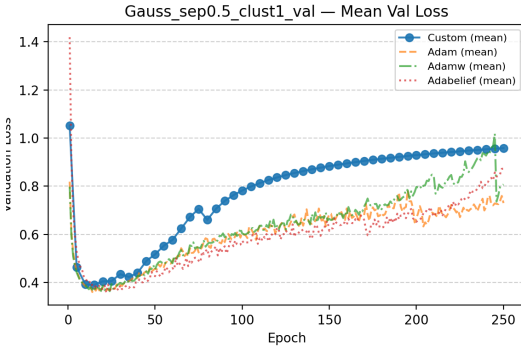

MNIST_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

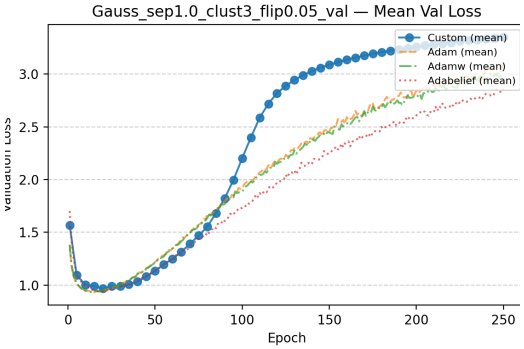

FashionMNIST_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

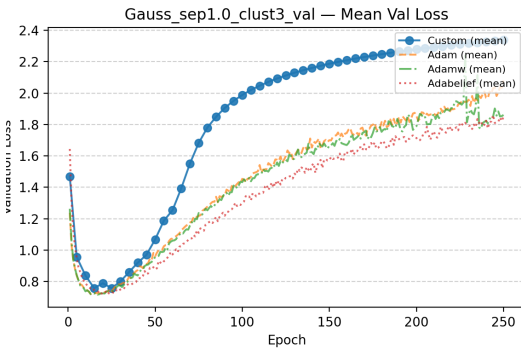

HAR_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

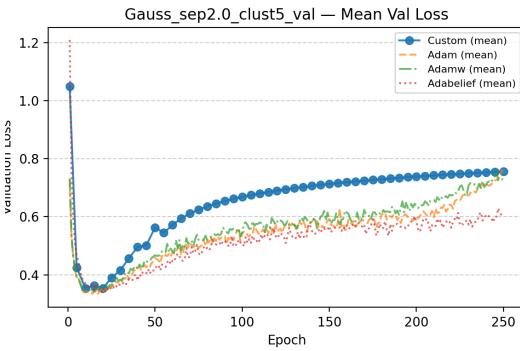

Imbalance_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch


WineQuality_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch


Gauss_sep0.5_clust1_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch


Gauss_sep1.0_clust3_flip0.05_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch


Gauss_sep1.0_clust3_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch


Gauss_sep2.0_clust5_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

E.2 Hidden7 With Regularization

TRAIN Results

dataset	custom (±sd)	adam (±sd)	adamw (±sd)	adabelief (±sd)	d_c_vs_a	d_c_vs_aw	d_c_vs_ab	p_ttest_c_vs_a	p_ttest_c_vs_aw	p_ttest_c_vs_ab	p_ttest_a_vs_aw	p_ttest_a_vs_ab	p_ttest_aw_vs_ab
MNIST	0.0247 ± 0.0013	0.1612 ± 0.0005	0.1609 ± 0.0004	0.1608 ± 0.0003	-137.4832	-142.3431	-145.0940	0.0000	0.0000	0.0000	0.1267	0.2365	0.9261
CIFAR10	0.7429 ± 0.0358	0.6283 ± 0.0058	0.6327 ± 0.0043	0.6257 ± 0.0051	4.4669	4.3190	4.5811	0.0001	0.0003	0.0001	0.2332	0.3800	0.0054
CIFAR100	2.5413 ± 0.0339	2.1177 ± 0.0093	2.1199 ± 0.0090	2.1065 ± 0.0045	17.0171	16.9728	17.9634	0.0000	0.0000	0.0000	0.6930	0.0521	0.0117
20NG	0.1198 ± 0.0044	0.2794 ± 0.0022	0.2786 ± 0.0018	0.2798 ± 0.0019	-45.7579	-47.0248	-46.7261	0.0000	0.0000	0.0000	0.2816	0.7298	0.3081
Imbalance	0.0237 ± 0.0026	0.1272 ± 0.0024	0.1274 ± 0.0015	0.1285 ± 0.0017	-41.5153	-48.7101	-47.2325	0.0000	0.0000	0.0000	0.7157	0.0892	0.1075
WineQuality	0.0454 ± 0.0167	0.1936 ± 0.0096	0.1882 ± 0.0068	0.2046 ± 0.0079	-10.8753	-11.1981	-12.1948	0.0000	0.0000	0.0000	0.1046	0.0072	0.0018
FashionMNIST	0.0529 ± 0.0055	0.1969 ± 0.0016	0.1980 ± 0.0017	0.1971 ± 0.0016	-35.2898	-35.3752	-35.3329	0.0000	0.0000	0.0000	0.3131	0.8176	0.3232
HAR	0.0865 ± 0.0178	0.1746 ± 0.0043	0.1730 ± 0.0026	0.1753 ± 0.0042	-6.8059	-6.8033	-6.8699	0.0000	0.0000	0.0000	0.4865	0.6997	0.2808
Gauss_sep0.5_clust1	0.0353 ± 0.0078	0.2123 ± 0.0155	0.2118 ± 0.0130	0.2112 ± 0.0139	-14.4376	-16.4366	-15.6166	0.0000	0.0000	0.0000	0.8796	0.7452	0.7496
Gauss_sep1.0_clust3	0.0800 ± 0.0149	0.3568 ± 0.0140	0.3571 ± 0.0113	0.3572 ± 0.0106	-19.1802	-20.9936	-21.4800	0.0000	0.0000	0.0000	0.9113	0.8822	0.9365
Gauss_sep2.0_clust5	0.0318 ± 0.0040	0.2043 ± 0.0063	0.2041 ± 0.0061	0.2050 ± 0.0071	-32.7129	-33.2350	-30.1852	0.0000	0.0000	0.0000	0.7092	0.7055	0.4967
Gauss_sep1.0_clust3_flip0.05	0.1433 ± 0.0061	0.4851 ± 0.0109	0.4850 ± 0.0165	0.4868 ± 0.0143	-38.8360	-27.5374	-31.3080	0.0000	0.0000	0.0000	0.9689	0.4600	0.3235

VAL Results

dataset	custom (±sd)	adam (±sd)	adamw (±sd)	adabelief (±sd)	d_c_vs_a	d_c_vs_aw	d_c_vs_ab	p_ttest_c_vs_a	p_ttest_c_vs_aw	p_ttest_c_vs_ab	p_ttest_a_vs_aw	p_ttest_a_vs_ab	p_ttest_aw_vs_ab
MNIST	0.0971 ± 0.0054	0.2358 ± 0.0050	0.2348 ± 0.0016	0.2359 ± 0.0057	-26.8527	-34.7578	-25.0900	0.0000	0.0000	0.0000	0.6600	0.9625	0.5541
CIFAR10	1.6472 ± 0.0500	2.1045 ± 0.0392	2.0711 ± 0.0620	2.0737 ± 0.0442	-10.1742	-7.5224	-9.0391	0.0000	0.0000	0.0000	0.3043	0.0672	0.9298
CIFAR100	3.3990 ± 0.0244	4.1294 ± 0.0386	4.1691 ± 0.1282	4.0858 ± 0.0203	-22.6393	-8.3457	-30.6258	0.0000	0.0000	0.0000	0.5099	0.0059	0.1731
20NG	2.6117 ± 0.1015	2.4017 ± 0.0340	2.4243 ± 0.0555	2.4122 ± 0.0331	2.7748	2.2916	2.6434	0.0022	0.0007	0.0006	0.3401	0.6446	0.4984
Imbalance	0.1378 ± 0.0141	0.2283 ± 0.0126	0.2258 ± 0.0108	0.2268 ± 0.0119	-6.7576	-7.0102	-6.8124	0.0000	0.0000	0.0000	0.3081	0.6342	0.7454
WineQuality	1.4845 ± 0.0949	1.4565 ± 0.1039	1.4934 ± 0.1438	1.4861 ± 0.0967	0.2821	-0.0729	-0.0161	0.2752	0.8270	0.9523	0.3108	0.4156	0.8398
FashionMNIST	0.4764 ± 0.0098	0.6230 ± 0.0163	0.6192 ± 0.0207	0.6153 ± 0.0118	-10.8971	-8.8080	-12.8013	0.0000	0.0000	0.0000	0.7679	0.1703	0.7195
HAR	0.1630 ± 0.0281	0.3083 ± 0.0459	0.2765 ± 0.0187	0.3631 ± 0.1665	-3.8163	-4.7600	-1.6762	0.0012	0.0001	0.0180	0.1420	0.4071	0.1792
Gauss_sep0.5_clust1	0.2300 ± 0.0471	0.3437 ± 0.0392	0.3373 ± 0.0315	0.3378 ± 0.0375	-2.6257	-2.6811	-2.5333	0.0000	0.0000	0.0000	0.1405	0.3320	0.9221
Gauss_sep1.0_clust3	0.6899 ± 0.0451	0.6601 ± 0.0205	0.6595 ± 0.0245	0.6531 ± 0.0292	0.8491	0.8374	0.9691	0.0347	0.0549	0.0142	0.8849	0.3124	0.2992
Gauss_sep2.0_clust5	0.2669 ± 0.0209	0.3318 ± 0.0120	0.3329 ± 0.0125	0.3315 ± 0.0159	-3.8098	-3.8302	-3.4846	0.0000	0.0000	0.0000	0.5368	0.9346	0.6696
Gauss_sep1.0_clust3_flip0.05	1.0385 ± 0.0389	0.9441 ± 0.0334	0.9517 ± 0.0294	0.9501 ± 0.0287	2.6036	2.5167	2.5823	0.0000	0.0001	0.0001	0.0632	0.3756	0.6657

ACC Results

dataset	custom (±sd)	adam (±sd)	adamw (±sd)	adabelief (±sd)	d_c_vs_a	d_c_vs_aw	d_c_vs_ab	p_ttest_c_vs_a	p_ttest_c_vs_aw	p_ttest_c_vs_ab	p_ttest_a_vs_aw	p_ttest_a_vs_ab	p_ttest_aw_vs_ab
MNIST	0.9800 ± 0.0008	0.9802 ± 0.0012	0.9799 ± 0.0006	0.9798 ± 0.0011	-0.2809	0.0396	0.1160	0.6667	0.9180	0.8254	0.5330	0.5506	0.8419
CIFAR10	0.5027 ± 0.0071	0.4641 ± 0.0065	0.4656 ± 0.0123	0.4698 ± 0.0081	5.6702	3.6800	4.3022	0.0000	0.0003	0.0000	0.7964	0.1962	0.5142
CIFAR100	0.2105 ± 0.0062	0.1898 ± 0.0049	0.1868 ± 0.0137	0.1961 ± 0.0041	3.7047	2.2222	2.7240	0.0004	0.0055	0.0008	0.6435	0.0203	0.2033
20NG	0.4988 ± 0.0173	0.5453 ± 0.0034	0.5427 ± 0.0096	0.5410 ± 0.0042	-3.7390	-3.1387	-3.3618	0.0004	0.0004	0.0003	0.4168	0.1056	0.6736
Imbalance	0.9659 ± 0.0039	0.9676 ± 0.0036	0.9683 ± 0.0030	0.9678 ± 0.0033	-0.4466	-0.6804	-0.5254	0.0633	0.0362	0.1646	0.4015	0.7944	0.6586
WineQuality	0.6518 ± 0.0202	0.6638 ± 0.0207	0.6594 ± 0.0250	0.6545 ± 0.0313	-0.5897	-0.3342	-0.1018	0.0814	0.4705	0.8137	0.4371	0.1887	0.4941
FashionMNIST	0.8862 ± 0.0014	0.8795 ± 0.0042	0.8803 ± 0.0040	0.8806 ± 0.0029	2.1623	1.9623	2.4680	0.0048	0.0042	0.0013	0.7690	0.3426	0.8941
HAR	0.9440 ± 0.0087	0.9458 ± 0.0160	0.9499 ± 0.0074	0.9337 ± 0.0258	-0.1345	-0.7293	0.5372	0.8507	0.3070	0.3802	0.4251	0.2090	0.0802
Gauss_sep0.5_clust1	0.9354 ± 0.0114	0.9327 ± 0.0131	0.9353 ± 0.0127	0.9347 ± 0.0122	0.2188	0.0079	0.0543	0.2287	0.9397	0.7185	0.1334	0.3229	0.6960
Gauss_sep1.0_clust3	0.8039 ± 0.0088	0.8289 ± 0.0075	0.8289 ± 0.0081	0.8295 ± 0.0097	-3.0487	-2.9510	-2.7561	0.0000	0.0001	0.0000	0.9848	0.7817	0.8195
Gauss_sep2.0_clust5	0.9243 ± 0.0073	0.9411 ± 0.0048	0.9403 ± 0.0041	0.9406 ± 0.0062	-2.7464	-2.7074	-2.4205	0.0000	0.0000	0.0003	0.0890	0.6992	0.8422
Gauss_sep1.0_clust3_flip0.05	0.7459 ± 0.0068	0.7689 ± 0.0096	0.7706 ± 0.0084	0.7691 ± 0.0080	-2.7585	-3.2368	-3.1196	0.0002	0.0001	0.0003	0.3008	0.9314	0.4158

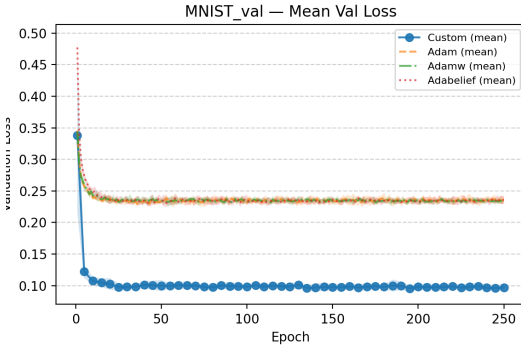
F1 Results

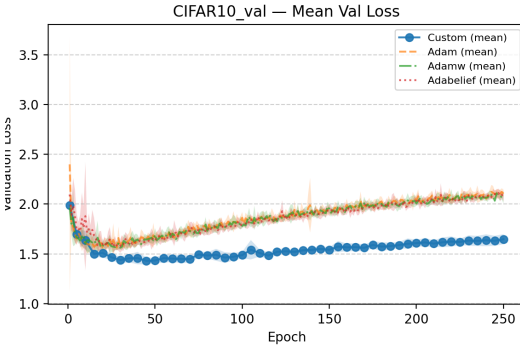
dataset	custom (±sd)	adam (±sd)	adamw (±sd)	adabelief (±sd)	d_c_vs_a	d_c_vs_aw	d_c_vs_ab	p_ttest_c_vs_a	p_ttest_c_vs_aw	p_ttest_c_vs_ab	p_ttest_a_vs_aw	p_ttest_a_vs_ab	p_ttest_aw_vs_ab
MNIST	0.9798 ± 0.0008	0.9801 ± 0.0012	0.9798 ± 0.0007	0.9797 ± 0.0012	-0.2812	0.0072	0.1210	0.6661	0.9849	0.8175	0.5659	0.5503	0.7949
CIFAR10	0.5027 ± 0.0072	0.4636 ± 0.0062	0.4662 ± 0.0124	0.4688 ± 0.0080	5.8177	3.6111	4.4457	0.0000	0.0003	0.0000	0.6564	0.1964	0.6633
CIFAR100	0.2012 ± 0.0054	0.1877 ± 0.0049	0.1829 ± 0.0122	0.1913 ± 0.0049	2.6143	1.9432	1.9319	0.0031	0.0097	0.0026	0.3878	0.2205	0.2144
20NG	0.4952 ± 0.0171	0.5391 ± 0.0039	0.5362 ± 0.0091	0.5346 ± 0.0046	-3.5498	-2.9965	-3.1486	0.0005	0.0005	0.0003	0.2744	0.0684	0.6014
Imbalance	0.9302 ± 0.0064	0.9317 ± 0.0086	0.9335 ± 0.0065	0.9319 ± 0.0078	-0.1987	-0.5098	-0.2345	0.4343	0.1147	0.4673	0.3808	0.8701	0.4318
WineQuality	0.4146 ± 0.0384	0.4265 ± 0.0336	0.4345 ± 0.0660	0.4107 ± 0.0404	-0.3314	-0.3701	0.0989	0.2425	0.3061	0.8163	0.6163	0.2983	0.3978
FashionMNIST	0.8862 ± 0.0013	0.8795 ± 0.0036	0.8800 ± 0.0044	0.8804 ± 0.0026	2.5040	1.9178	2.8281	0.0018	0.0058	0.0005	0.8434	0.3579	0.8664
HAR	0.9446 ± 0.0087	0.9474 ± 0.0150	0.9516 ± 0.0070	0.9345 ± 0.0278	-0.2340	-0.8933	0.4910	0.7457	0.2172	0.4162	0.4003	0.2085	0.0955
Gauss_sep0.5_clust1	0.9353 ± 0.0114	0.9325 ± 0.0131	0.9351 ± 0.0127	0.9346 ± 0.0123	0.2247	0.0151	0.0616	0.2194	0.8847	0.6836	0.1366	0.3282	0.6982
Gauss_sep1.0_clust3	0.8039 ± 0.0089	0.8288 ± 0.0075	0.8288 ± 0.0081	0.8294 ± 0.0099	-3.0233	-2.9292	-2.7141	0.0000	0.0001	0.0000	0.9750	0.7782	0.8217
Gauss_sep2.0_clust5	0.9243 ± 0.0073	0.9411 ± 0.0048	0.9403 ± 0.0041	0.9406 ± 0.0062	-2.7477	-2.7114	-2.4168	0.0000	0.0000	0.0003	0.0940	0.6972	0.8532
Gauss_sep1.0_clust3_flip0.05	0.7457 ± 0.0068	0.7688 ± 0.0097	0.7706 ± 0.0084	0.7690 ± 0.0080	-2.7621	-3.2432	-3.1391	0.0002	0.0001	0.0003	0.2827	0.9344	0.4021

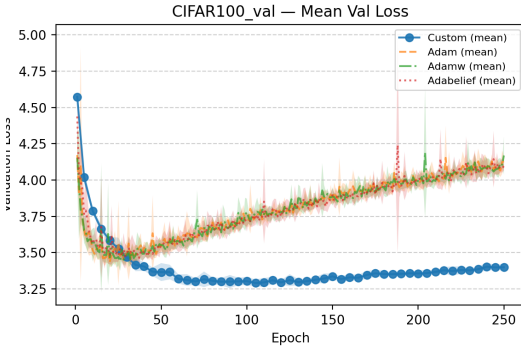
TIME Results

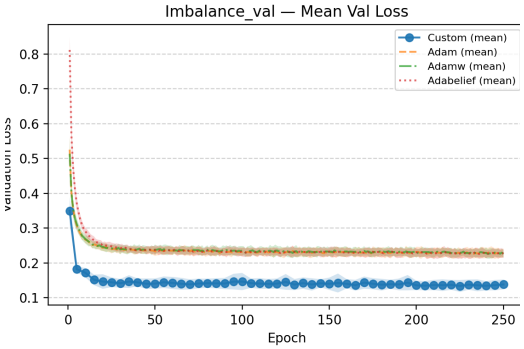
dataset	custom (±sd)	adam (±sd)	adamw (±sd)	adabelief (±sd)
MNIST	147.3702 ± 2.9565	324.4420 ± 15.4161	327.3905 ± 7.4020	333.9292 ± 6.3580
CIFAR10	167.0759 ± 3.2113	305.4651 ± 8.6221	313.3427 ± 9.8854	322.6399 ± 13.1202
CIFAR100	179.0844 ± 6.1374	314.2856 ± 18.2871	324.2971 ± 16.9202	329.9635 ± 16.9840
20NG	48.0550 ± 1.2739	132.2616 ± 22.3313	140.8964 ± 15.7087	144.6984 ± 11.5703
Imbalance	56.1017 ± 1.2223	144.5799 ± 2.4042	147.8887 ± 2.5363	150.7667 ± 2.5154
WineQuality	8.6256 ± 0.6152	22.8293 ± 0.4181	23.5176 ± 0.3628	25.5559 ± 0.5273
FashionMNIST	158.2560 ± 1.8430	341.5315 ± 10.4747	350.6788 ± 10.0860	357.6487 ± 10.5137
HAR	27.4807 ± 0.8026	88.2865 ± 10.9943	92.9363 ± 4.5350	98.0614 ± 0.5326
Gauss_sep0.5_clust1	57.7546 ± 0.8784	147.6593 ± 0.7899	150.6592 ± 0.8316	153.8902 ± 1.2198
Gauss_sep1.0_clust3	57.0510 ± 0.9179	140.0073 ± 4.2219	144.1197 ± 2.9010	146.7665 ± 2.6952
Gauss_sep2.0_clust5	57.3562 ± 1.5548	143.5710 ± 2.0714	147.0475 ± 2.1150	150.7276 ± 2.6865
Gauss_sep1.0_clust3_flip0.05	57.3396 ± 0.7154	144.8166 ± 2.6666	146.2454 ± 1.4748	149.9836 ± 3.3339

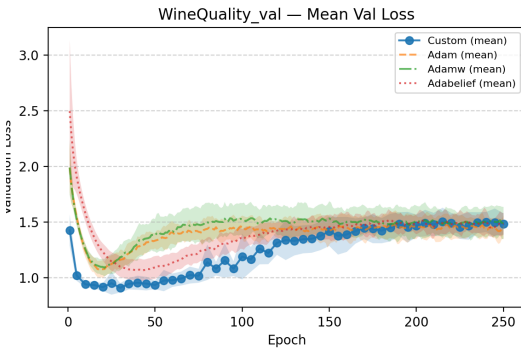
Figure 2: Summary of Experimental Results for Hidden7 With Regularization

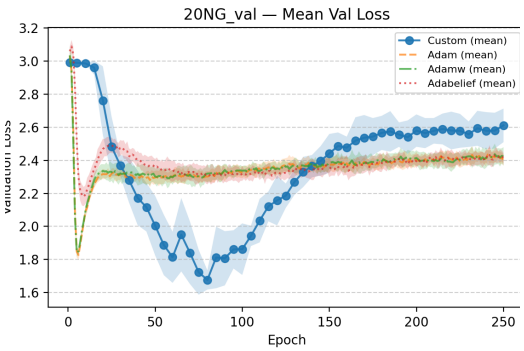

MNIST_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

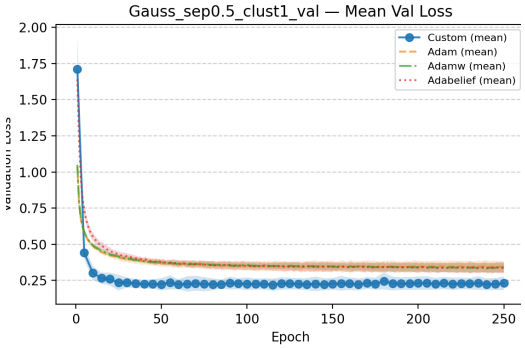

CIFAR10_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

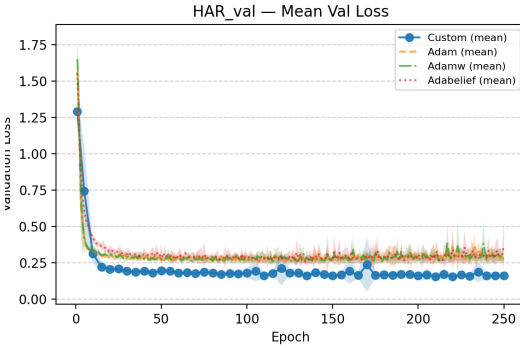

CIFAR100_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

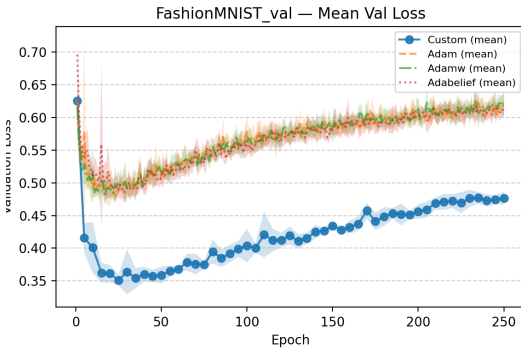

Imbalance_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

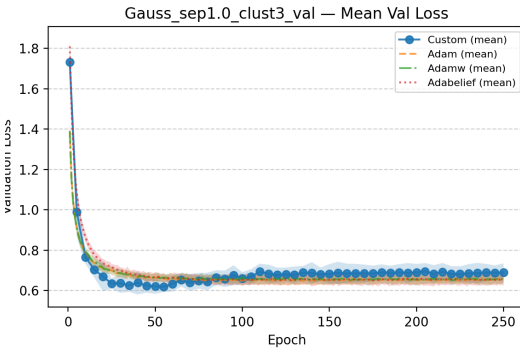

WineQuality_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

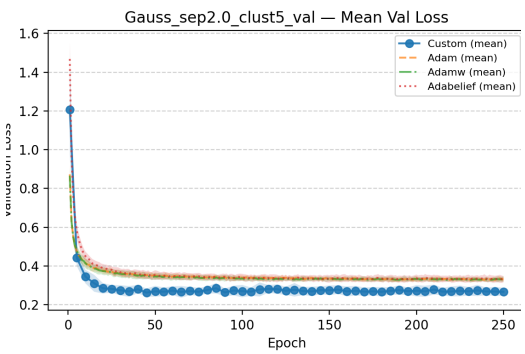

20NG_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

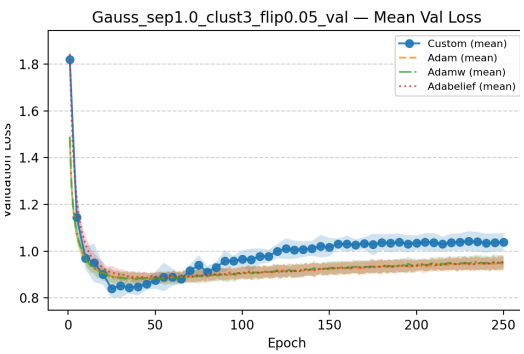

Gauss_sep0.5_clust1_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch


HAR_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch


FashionMNIST_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch


Gauss_sep1.0_clust3_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch


Gauss_sep2.0_clust5_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch


Gauss_sep1.0_clust3_flip0.05_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

E.3 Hidden10 Without Regularization

TRAIN Results

	Custom	Adam	Adamw	Adabelief	p_c_vs_a	p_c_vs_aw	p_c_vs_ab
MNIST	0.0	0.0015	0.0008	0.0039	0.1503	0.1905	0.0119
CIFAR10	0.7543	0.8545	0.8521	0.7896	0.0157	0.0402	0.3134
CIFAR100	2.5776	2.4628	2.4762	2.4193	0.1339	0.1674	0.0245
20NG	0.0904	0.0934	0.0942	0.1018	0.0335	0.0258	0.1874
Imbalance	0.0	0.0042	0.0067	0.0053	0.0573	0.0405	0.0206
WineQuality	0.0025	0.0117	0.002	0.0255	0.2191	0.5287	0.3515
FashionMNIST	0.0244	0.094	0.05	0.0424	0.152	0.0015	0.0014
HAR	0.056	0.0239	0.0281	0.033	0.0008	0.0007	0.0185
Gauss_sep0.5_clust1	0.0001	0.0299	0.0346	0.0262	0.0002	0.0003	0.0004
Gauss_sep1.0_clust3	0.0001	0.0906	0.0706	0.0811	0.0001	0.0	0.0
Gauss_sep2.0_clust5	0.0	0.0279	0.027	0.0259	0.0	0.0	0.0001
Gauss_sep1.0_clust3_flip0.05	0.0002	0.1337	0.1214	0.1429	0.0	0.0	0.0

VAL Results

	Custom	Adam	Adamw	Adabelief	p_c_vs_a	p_c_vs_aw	p_c_vs_ab
MNIST	0.2457	0.3456	0.4691	0.29	0.125	0.027	0.2694
CIFAR10	2.4471	2.0845	2.0848	2.0761	0.014	0.0116	0.0057
CIFAR100	3.9646	4.1963	4.2123	4.0919	0.001	0.0005	0.003
20NG	5.9258	5.0125	4.7149	4.7243	0.0001	0.0001	0.0058
Imbalance	0.5237	0.5612	0.4588	0.5155	0.7219	0.3399	0.9105
WineQuality	3.0536	3.033	3.5869	3.2316	0.9301	0.0028	0.5807
FashionMNIST	1.1178	0.7771	0.7757	0.8454	0.0002	0.0003	0.0004
HAR	0.263	0.2584	0.2181	0.2081	0.8039	0.0503	0.0014
Gauss_sep0.5_clust1	1.1794	0.7212	0.7165	0.7206	0.0	0.0	0.0
Gauss_sep1.0_clust3	2.8179	1.6139	1.6025	1.5534	0.0	0.0	0.0
Gauss_sep2.0_clust5	0.9305	0.545	0.5394	0.5335	0.0	0.0	0.0
Gauss_sep1.0_clust3_flip0.05	3.7196	2.4249	2.4281	2.2781	0.0	0.0	0.0

ACC Results

	Custom	Adam	Adamw	Adabelief	p_c_vs_a	p_c_vs_aw	p_c_vs_ab
MNIST	0.9761	0.9791	0.978	0.9777	0.0024	0.2404	0.0712
CIFAR10	0.4797	0.459	0.4566	0.4701	0.0004	0.0023	0.0443
CIFAR100	0.1793	0.1783	0.1758	0.1827	0.7893	0.1937	0.1322
20NG	0.5365	0.5386	0.5465	0.5377	0.6914	0.045	0.6854
Imbalance	0.9553	0.9564	0.9556	0.9548	0.5305	0.8516	0.8137
WineQuality	0.6491	0.6625	0.6402	0.6554	0.1342	0.3962	0.7043
FashionMNIST	0.8762	0.8755	0.8822	0.8842	0.9146	0.0724	0.0028
HAR	0.9383	0.9535	0.9551	0.9545	0.0025	0.0005	0.0035
Gauss_sep0.5_clust1	0.8995	0.8866	0.8841	0.8867	0.0071	0.0007	0.0059
Gauss_sep1.0_clust3	0.7767	0.7631	0.7685	0.7662	0.0094	0.0312	0.0245
Gauss_sep2.0_clust5	0.9139	0.9119	0.9122	0.9145	0.2571	0.4062	0.7353
Gauss_sep1.0_clust3_flip0.05	0.7183	0.6932	0.6927	0.6936	0.0002	0.0003	0.0001

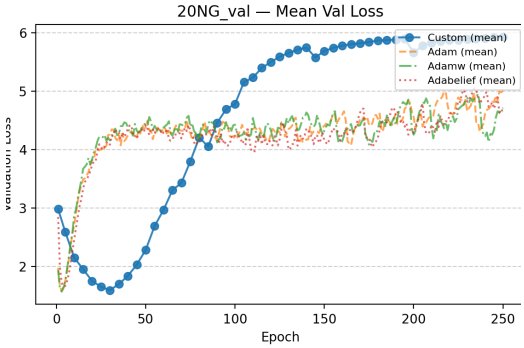
F1 Results

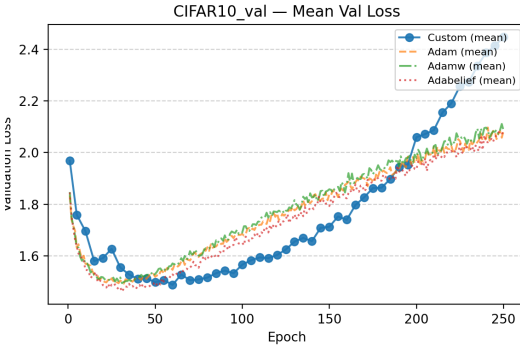
	Custom	Adam	Adamw	Adabelief	p_c_vs_a	p_c_vs_aw	p_c_vs_ab
MNIST	0.9759	0.9789	0.9778	0.9775	0.0028	0.2425	0.0729
CIFAR10	0.479	0.4581	0.4551	0.4687	0.0004	0.0019	0.031
CIFAR100	0.1716	0.1727	0.1699	0.1763	0.7798	0.5497	0.0594
20NG	0.5316	0.5354	0.5427	0.5334	0.4527	0.0283	0.4643
Imbalance	0.9076	0.9121	0.9094	0.9082	0.1834	0.5076	0.8827
WineQuality	0.4209	0.4195	0.4184	0.4236	0.9257	0.8962	0.8686
FashionMNIST	0.8762	0.8754	0.882	0.8844	0.9041	0.087	0.0025
HAR	0.938	0.9538	0.9558	0.9552	0.0026	0.0004	0.0029
Gauss_sep0.5_clust1	0.8995	0.8865	0.8841	0.8867	0.0073	0.0007	0.0061
Gauss_sep1.0_clust3	0.7767	0.7629	0.7684	0.7662	0.0089	0.0269	0.0236
Gauss_sep2.0_clust5	0.9139	0.9119	0.9122	0.9145	0.2643	0.4142	0.7268
Gauss_sep1.0_clust3_flip0.05	0.7181	0.693	0.6924	0.6935	0.0003	0.0002	0.0001

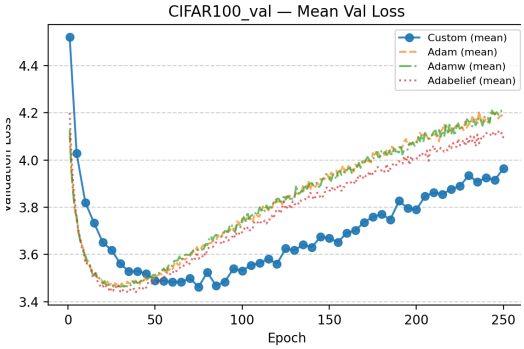
TIME Results

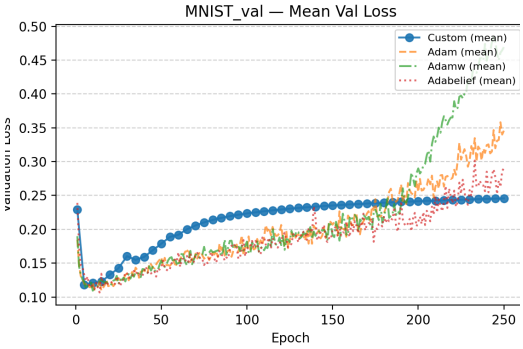
	Custom	Adam	Adamw	Adabelief
MNIST	178.7493	275.1787	284.1111	290.527
CIFAR10	190.9793	277.2164	283.7629	307.2627
CIFAR100	207.4634	282.5955	291.8332	312.1756
20NG	59.46	92.9509	97.325	100.7718
Imbalance	65.5428	118.4874	122.7422	124.601
WineQuality	8.2637	13.4492	16.511	17.7939
FashionMNIST	179.1427	281.9967	292.2724	297.1215
HAR	29.25	51.7072	53.9491	57.4662
Gauss_sep0.5_clust1	64.7157	117.0886	120.0175	122.2473
Gauss_sep1.0_clust3	64.5793	120.7696	122.3737	123.0598
Gauss_sep2.0_clust5	64.151	119.9376	122.3609	123.4184
Gauss_sep1.0_clust3_flip0.05	63.5085	118.705	120.8227	122.9597

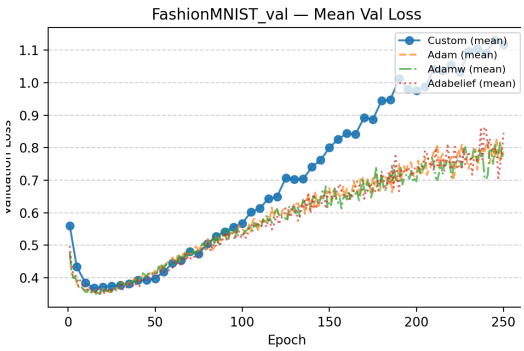
Figure 3: Summary of Experimental Results for Hidden10 Without Regularization

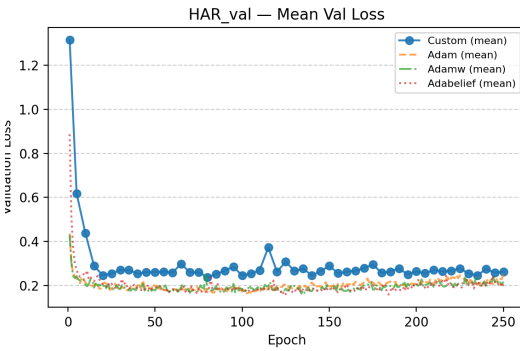

20NG_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

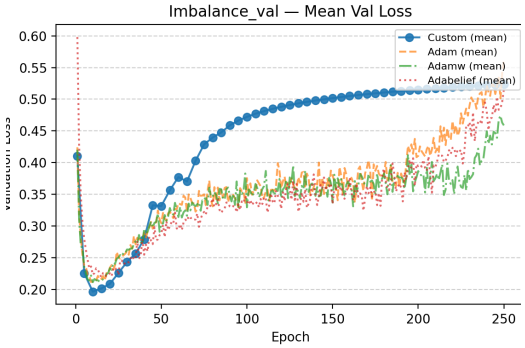

CIFAR10_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

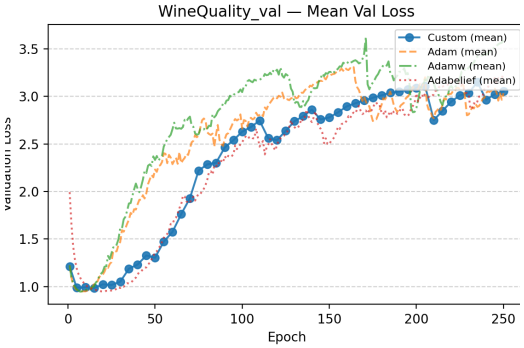

CIFAR100_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

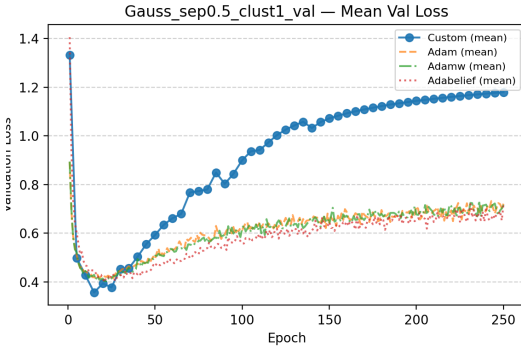

MNIST_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

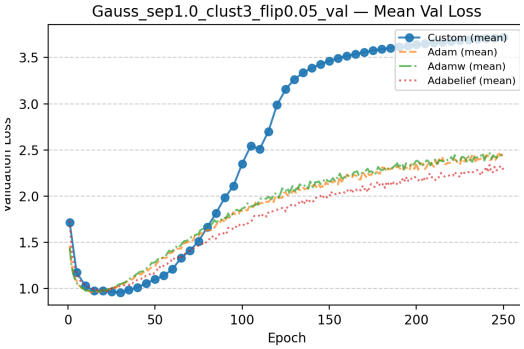

FashionMNIST_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

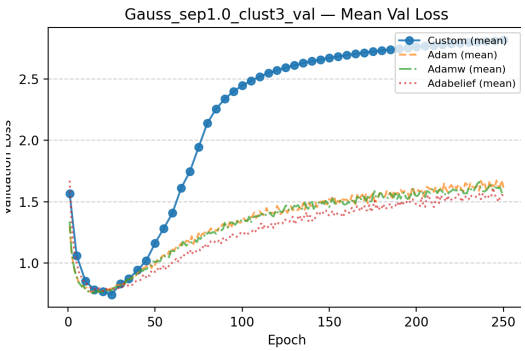

HAR_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

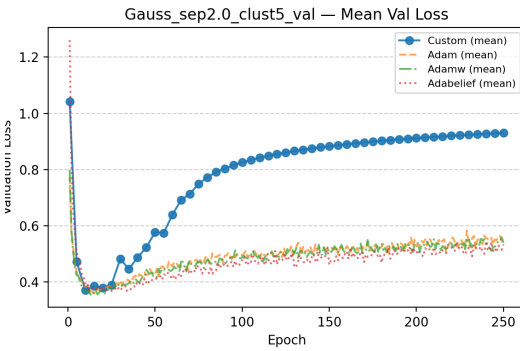

Imbalance_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch


WineQuality_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch


Gauss_sep0.5_clust1_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch


Gauss_sep1.0_clust3_flip0.05_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch


Gauss_sep1.0_clust3_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch


Gauss_sep2.0_clust5_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

E.4 Hidden10 With Regularization

TRAIN Results

dataset	custom (±sd)	adam (±sd)	adamw (±sd)	adabelief (±sd)	d_c_vs_a	d_c_vs_aw	d_c_vs_ab	p_ttest_c_vs_a	p_ttest_c_vs_aw	p_ttest_c_vs_ab	p_ttest_a_vs_aw	p_ttest_a_vs_ab	p_ttest_aw_vs_ab
MNIST	0.0289 ± 0.0078	0.1616 ± 0.0007	0.1616 ± 0.0005	0.1620 ± 0.0006	-23.8963	-23.9367	-23.9840	0.0000	0.0000	0.0000	0.7908	0.1815	0.2220
CIFAR10	0.8498 ± 0.0313	0.6565 ± 0.0078	0.6546 ± 0.0055	0.6540 ± 0.0046	8.4787	8.6878	8.7577	0.0000	0.0000	0.0000	0.6072	0.3549	0.6103
CIFAR100	2.8258 ± 0.0541	2.2222 ± 0.0058	2.2212 ± 0.0080	2.2092 ± 0.0057	15.6940	15.6410	16.0349	0.0000	0.0000	0.0000	0.7897	0.0035	0.0103
20NG	0.1651 ± 0.0360	0.2858 ± 0.0015	0.2857 ± 0.0040	0.2874 ± 0.0018	-4.7295	-4.7023	-4.7923	0.0001	0.0001	0.0001	0.9779	0.0412	0.3160
Imbalance	0.0264 ± 0.0071	0.1288 ± 0.0025	0.1288 ± 0.0021	0.1295 ± 0.0026	-19.1598	-19.4729	-19.1675	0.0000	0.0000	0.0000	0.9528	0.2814	0.4535
WineQuality	0.0484 ± 0.0125	0.2045 ± 0.0105	0.2019 ± 0.0103	0.2216 ± 0.0115	-13.5559	-13.4090	-14.4118	0.0000	0.0000	0.0000	0.6164	0.0207	0.0028
FashionMNIST	0.0737 ± 0.0090	0.2006 ± 0.0022	0.2019 ± 0.0009	0.2008 ± 0.0019	-19.3428	-19.9826	-19.4849	0.0000	0.0000	0.0000	0.1846	0.8894	0.2842
HAR	0.0825 ± 0.0128	0.1832 ± 0.0041	0.1852 ± 0.0017	0.1856 ± 0.0040	-10.5530	-11.2100	-10.8457	0.0000	0.0000	0.0000	0.3311	0.0965	0.8099
Gauss_sep0.5_ clust1	0.0419 ± 0.0106	0.2178 ± 0.0158	0.2157 ± 0.0118	0.2151 ± 0.0115	-13.0821	-15.5027	-15.6837	0.0000	0.0000	0.0000	0.4627	0.4749	0.7862
Gauss_sep1.0_ clust3	0.0901 ± 0.0063	0.3467 ± 0.0119	0.3446 ± 0.0086	0.3482 ± 0.0112	-27.0071	-33.8030	-28.3326	0.0000	0.0000	0.0000	0.3212	0.5273	0.1318
Gauss_sep2.0_ clust5	0.0323 ± 0.0033	0.2037 ± 0.0054	0.2036 ± 0.0050	0.2047 ± 0.0050	-38.3166	-40.1472	-40.7110	0.0000	0.0000	0.0000	0.8866	0.3208	0.3994
Gauss_sep1.0_ clust3_flip0.05	0.1423 ± 0.0103	0.4724 ± 0.0102	0.4716 ± 0.0144	0.4706 ± 0.0130	-32.2868	-26.3526	-28.0454	0.0000	0.0000	0.0000	0.7771	0.2783	0.6642

VAL Results

dataset	custom (±sd)	adam (±sd)	adamw (±sd)	adabelief (±sd)	d_c_vs_a	d_c_vs_aw	d_c_vs_ab	p_ttest_c_vs_a	p_ttest_c_vs_aw	p_ttest_c_vs_ab	p_ttest_a_vs_aw	p_ttest_a_vs_ab	p_ttest_aw_vs_ab
MNIST	0.1092 ± 0.0057	0.2332 ± 0.0034	0.2349 ± 0.0018	0.2352 ± 0.0049	-26.5773	-29.8988	-23.7693	0.0000	0.0000	0.0000	0.4075	0.2400	0.8819
CIFAR10	1.5079 ± 0.0502	2.0000 ± 0.0434	2.0023 ± 0.0601	2.0078 ± 0.0515	-10.4885	-8.9285	-9.8345	0.0000	0.0000	0.0000	0.9340	0.8062	0.7941
CIFAR100	3.4660 ± 0.0216	3.9369 ± 0.0250	3.9386 ± 0.0408	3.9587 ± 0.0494	-20.1333	-14.4769	-12.9123	0.0000	0.0000	0.0000	0.9107	0.3064	0.2538
20NG	2.8253 ± 0.0668	2.5061 ± 0.0439	2.5230 ± 0.0484	2.5019 ± 0.0630	5.6500	5.1840	4.9818	0.0000	0.0000	0.0000	0.3918	0.8342	0.4048
Imbalance	0.1375 ± 0.0165	0.2317 ± 0.0108	0.2321 ± 0.0177	0.2327 ± 0.0118	-6.7558	-5.5244	-6.6274	0.0000	0.0000	0.0000	0.9128	0.6461	0.8819
WineQuality	1.4310 ± 0.1263	1.4731 ± 0.1315	1.4813 ± 0.1438	1.4393 ± 0.0564	-0.3262	-0.3714	-0.0842	0.2406	0.0911	0.8319	0.8336	0.4136	0.3496
FashionMNIST	0.4528 ± 0.0107	0.6178 ± 0.0106	0.6108 ± 0.0170	0.6038 ± 0.0166	-15.4949	-11.1230	-10.7884	0.0000	0.0000	0.0000	0.4347	0.1516	0.3382
HAR	0.1608 ± 0.0168	0.2735 ± 0.0305	0.2785 ± 0.0409	0.2689 ± 0.0313	-4.5813	-3.7632	-4.3114	0.0001	0.0005	0.0001	0.8091	0.5319	0.6228
Gauss_sep0.5_ clust1	0.2397 ± 0.0359	0.3558 ± 0.0438	0.3569 ± 0.0310	0.3586 ± 0.0325	-2.9012	-3.4953	-3.4752	0.0000	0.0000	0.0000	0.8791	0.8034	0.7681
Gauss_sep1.0_ clust3	0.6662 ± 0.0405	0.6753 ± 0.0214	0.6672 ± 0.0204	0.6703 ± 0.0255	-0.2805	-0.0332	-0.1229	0.4751	0.9107	0.7501	0.1789	0.4973	0.6240
Gauss_sep2.0_ clust5	0.2789 ± 0.0254	0.3373 ± 0.0191	0.3418 ± 0.0156	0.3382 ± 0.0145	-2.5986	-2.9865	-2.8699	0.0000	0.0002	0.0000	0.2840	0.7476	0.3236
Gauss_sep1.0_ clust3_flip0.05	0.9767 ± 0.0388	0.9397 ± 0.0259	0.9416 ± 0.0286	0.9433 ± 0.0323	1.1215	1.0298	0.9363	0.0053	0.0086	0.0051	0.7797	0.5218	0.5710

ACC Results

dataset	custom (±sd)	adam (±sd)	adamw (±sd)	adabelief (±sd)	d_c_vs_a	d_c_vs_aw	d_c_vs_ab	p_ttest_c_vs_a	p_ttest_c_vs_aw	p_ttest_c_vs_ab	p_ttest_a_vs_aw	p_ttest_a_vs_ab	p_ttest_aw_vs_ab
MNIST	0.9791 ± 0.0011	0.9807 ± 0.0007	0.9801 ± 0.0006	0.9803 ± 0.0014	-1.7340	-1.1087	-0.9772	0.0017	0.1114	0.1654	0.1935	0.4562	0.7164
CIFAR10	0.5109 ± 0.0079	0.4677 ± 0.0049	0.4686 ± 0.0147	0.4680 ± 0.0088	6.5965	3.5812	5.1604	0.0000	0.0000	0.0000	0.8739	0.9573	0.9050
CIFAR100	0.1811 ± 0.0066	0.1996 ± 0.0053	0.2025 ± 0.0061	0.1989 ± 0.0038	-3.0743	-3.3724	-3.3092	0.0003	0.0002	0.0007	0.2890	0.7867	0.1285
20NG	0.4619 ± 0.0126	0.5407 ± 0.0102	0.5401 ± 0.0078	0.5359 ± 0.0085	-6.8915	-7.4772	-6.8756	0.0000	0.0000	0.0000	0.8850	0.0582	0.2836
Imbalance	0.9673 ± 0.0040	0.9665 ± 0.0031	0.9668 ± 0.0049	0.9664 ± 0.0025	0.2119	0.1009	0.2632	0.5061	0.8088	0.4764	0.7630	0.8452	0.7036
WineQuality	0.6589 ± 0.0279	0.6670 ± 0.0291	0.6509 ± 0.0224	0.6634 ± 0.0216	-0.2822	0.3181	-0.1790	0.3250	0.1755	0.6273	0.0569	0.3932	0.1194
FashionMNIST	0.8845 ± 0.0024	0.8797 ± 0.0037	0.8797 ± 0.0039	0.8815 ± 0.0043	1.5556	1.5039	0.8754	0.0447	0.0209	0.1213	0.9945	0.4574	0.5382
HAR	0.9460 ± 0.0078	0.9503 ± 0.0093	0.9442 ± 0.0152	0.9489 ± 0.0090	-0.5099	0.1437	-0.3476	0.3108	0.8134	0.4906	0.4074	0.5687	0.5153
Gauss_sep0.5_ clust1	0.9321 ± 0.0090	0.9302 ± 0.0149	0.9305 ± 0.0103	0.9289 ± 0.0101	0.1603	0.1719	0.3407	0.5226	0.3828	0.2438	0.9155	0.7655	0.4794
Gauss_sep1.0_ clust3	0.8049 ± 0.0103	0.8249 ± 0.0074	0.8282 ± 0.0062	0.8265 ± 0.0070	-2.2243	-2.7384	-2.4512	0.0013	0.0000	0.0001	0.2611	0.4892	0.1914
Gauss_sep2.0_ clust5	0.9216 ± 0.0076	0.9413 ± 0.0059	0.9385 ± 0.0063	0.9404 ± 0.0047	-2.8923	-2.4144	-2.9706	0.0000	0.0001	0.0000	0.1321	0.5534	0.2756
Gauss_sep1.0_ clust3_flip0.05	0.7500 ± 0.0086	0.7687 ± 0.0089	0.7685 ± 0.0107	0.7705 ± 0.0095	-2.1340	-1.8983	-2.2615	0.0000	0.0002	0.0000	0.8615	0.1923	0.1619

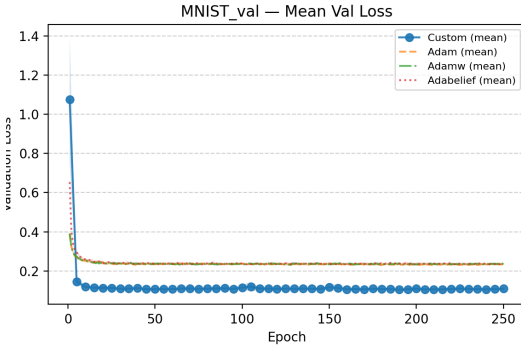
F1 Results

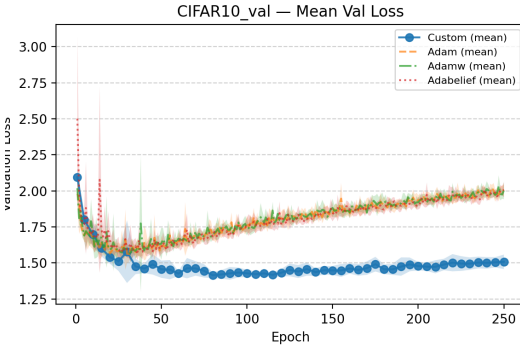
dataset	custom (±sd)	adam (±sd)	adamw (±sd)	adabelief (±sd)	d_c_vs_a	d_c_vs_aw	d_c_vs_ab	p_ttest_c_vs_a	p_ttest_c_vs_aw	p_ttest_c_vs_ab	p_ttest_a_vs_aw	p_ttest_a_vs_ab	p_ttest_aw_vs_ab
MNIST	0.9789 ± 0.0011	0.9805 ± 0.0007	0.9799 ± 0.0006	0.9802 ± 0.0014	-1.7549	-1.1700	-1.0179	0.0018	0.0987	0.1538	0.2190	0.5186	0.7059
CIFAR10	0.5106 ± 0.0077	0.4660 ± 0.0052	0.4681 ± 0.0154	0.4676 ± 0.0075	6.8069	3.4943	5.6740	0.0000	0.0000	0.0000	0.7272	0.7281	0.9264
CIFAR100	0.1674 ± 0.0051	0.1951 ± 0.0046	0.1972 ± 0.0053	0.1952 ± 0.0036	-5.6818	-5.7458	-6.2654	0.0000	0.0000	0.0000	0.3321	0.9771	0.2748
20NG	0.4592 ± 0.0127	0.5344 ± 0.0115	0.5348 ± 0.0073	0.5293 ± 0.0089	-6.2041	-7.3199	-6.4008	0.0001	0.0000	0.0001	0.9236	0.0989	0.1928
Imbalance	0.9324 ± 0.0067	0.9310 ± 0.0066	0.9304 ± 0.0094	0.9295 ± 0.0077	0.2085	0.2356	0.3977	0.4024	0.5379	0.2302	0.7443	0.2561	0.5868
WineQuality	0.4449 ± 0.0521	0.4250 ± 0.0346	0.4209 ± 0.0282	0.4224 ± 0.0310	0.4494	0.5734	0.5250	0.1639	0.1462	0.2091	0.3352	0.8271	0.8975
FashionMNIST	0.8846 ± 0.0025	0.8797 ± 0.0037	0.8792 ± 0.0041	0.8815 ± 0.0038	1.5776	1.5741	0.9638	0.0440	0.0225	0.1063	0.8536	0.4260	0.4322
HAR	0.9464 ± 0.0077	0.9516 ± 0.0092	0.9460 ± 0.0150	0.9506 ± 0.0085	-0.6225	0.0258	-0.5263	0.2344	0.9657	0.3127	0.4444	0.6934	0.5114
Gauss_sep0.5_ clust1	0.9321 ± 0.0090	0.9300 ± 0.0149	0.9304 ± 0.0104	0.9288 ± 0.0101	0.1652	0.1757	0.3458	0.5135	0.3800	0.2423	0.9096	0.7678	0.4778
Gauss_sep1.0_ clust3	0.8048 ± 0.0104	0.8247 ± 0.0072	0.8280 ± 0.0062	0.8264 ± 0.0071	-2.2253	-2.7103	-2.4248	0.0013	0.0000	0.0001	0.2593	0.5008	0.1800
Gauss_sep2.0_ clust5	0.9217 ± 0.0076	0.9413 ± 0.0059	0.9385 ± 0.0063	0.9404 ± 0.0047	-2.8841	-2.4127	-2.9633	0.0000	0.0001	0.0000	0.1337	0.5536	0.2815
Gauss_sep1.0_ clust3_flip0.05	0.7499 ± 0.0086	0.7685 ± 0.0090	0.7683 ± 0.0108	0.7704 ± 0.0097	-2.1058	-1.8826	-2.2418	0.0000	0.0002	0.0000	0.8811	0.1718	0.1419

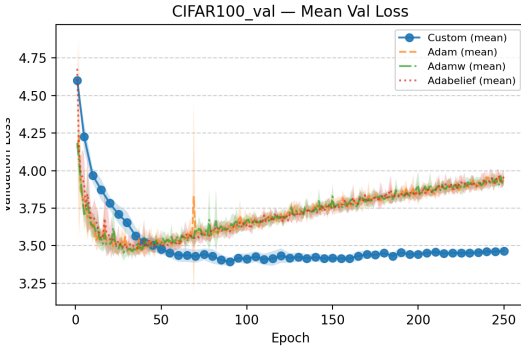
TIME Results

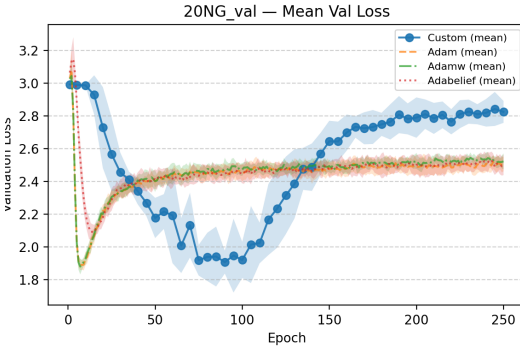
dataset	custom (±sd)	adam (±sd)	adamw (±sd)	adabelief (±sd)
MNIST	185.4826 ± 6.4292	376.7281 ± 11.4109	393.0337 ± 16.0912	398.2448 ± 12.3954
CIFAR10	191.2558 ± 5.6225	434.8170 ± 92.2056	503.9228 ± 54.0965	502.8245 ± 70.2491
CIFAR100	204.6625 ± 6.6011	503.2338 ± 69.7559	531.9348 ± 17.1434	539.6793 ± 18.0731
20NG	60.3703 ± 1.1462	165.5293 ± 29.4061	184.4452 ± 7.7035	185.8137 ± 6.6768
Imbalance	70.8542 ± 0.7904	265.2795 ± 1.7394	268.9805 ± 2.4659	273.2321 ± 2.7111
WineQuality	10.0686 ± 0.6547	28.0815 ± 0.6277	29.0738 ± 0.6801	31.7435 ± 0.4252
FashionMNIST	192.4882 ± 4.7716	411.8450 ± 7.0397	445.4972 ± 7.8580	458.7871 ± 56.3374
HAR	31.9890 ± 0.6137	101.8450 ± 2.5930	104.5518 ± 2.2058	108.2547 ± 2.1761
Gauss_sep0.5_ clust1	69.0256 ± 1.0791	224.3607 ± 44.4264	231.8459 ± 40.6072	256.6878 ± 4.1680
Gauss_sep1.0_ clust3	67.8650 ± 1.1989	230.7364 ± 26.7566	227.6278 ± 40.9214	246.7569 ± 24.6152
Gauss_sep2.0_ clust5	65.8892 ± 0.9686	209.2319 ± 41.6651	222.3391 ± 31.7962	225.3823 ± 41.3267
Gauss_sep1.0_ clust3_flip0.05	66.6435 ± 1.1881	224.5177 ± 25.2192	225.7552 ± 28.9031	238.8763 ± 16.2463

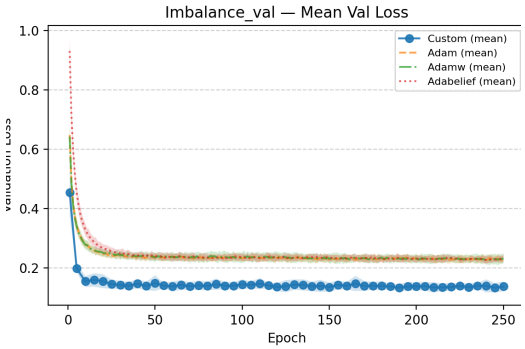
Figure 4: Summary of Experimental Results for Hidden10 With Regularization

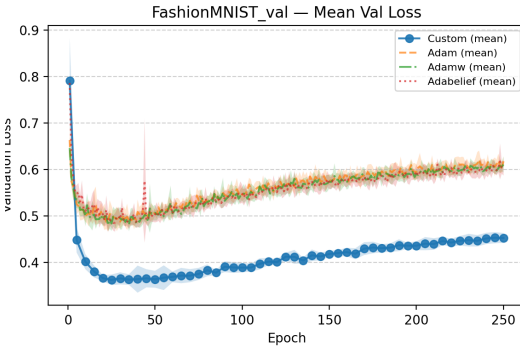

MNIST_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

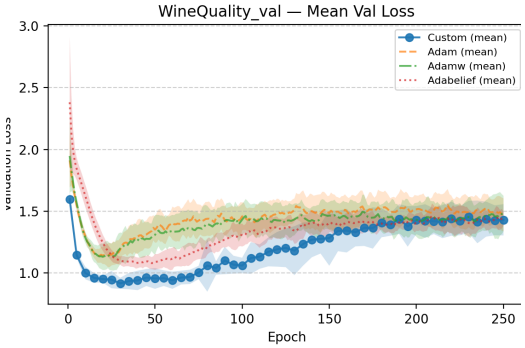

CIFAR10_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

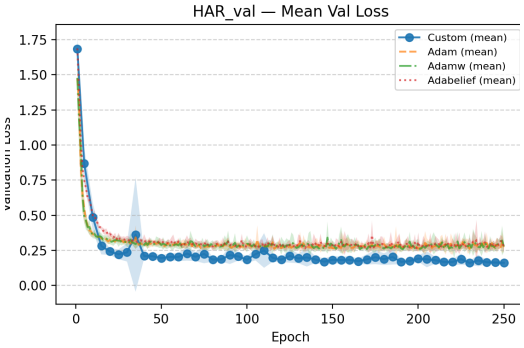

CIFAR100_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

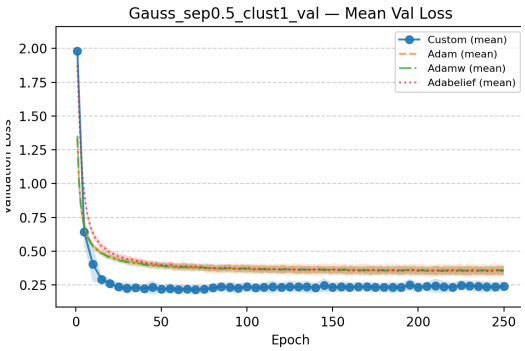

20NG_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

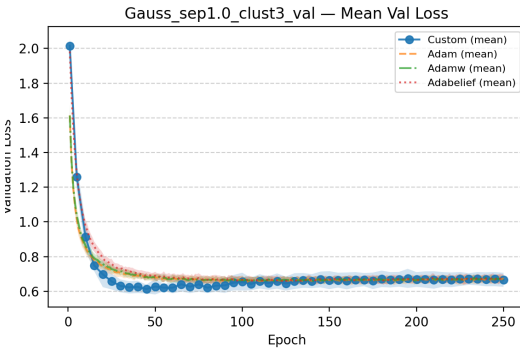

Imbalance_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

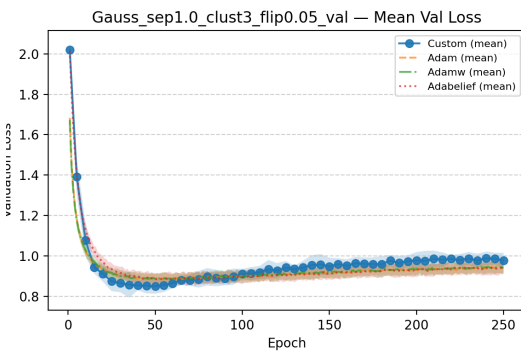

FashionMNIST_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

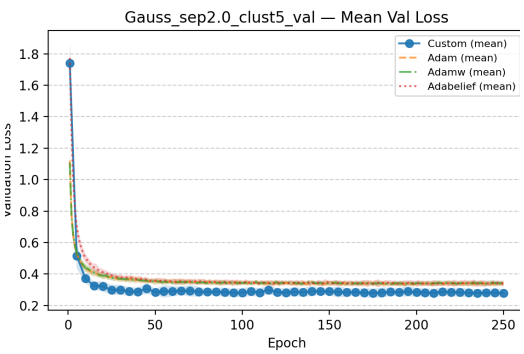

WineQuality_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch


HAR_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch


Gauss_sep0.5_clust1_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch


Gauss_sep1.0_clust3_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch


Gauss_sep1.0_clust3_flip0.05_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch


Gauss_sep2.0_clust5_val — Mean Val Loss
Custom (mean)
Adam (mean)
Adamw (mean)
Adabelief (mean)
Epoch

E.5 Hyperparameter Sensitivity Analysis Heatmap

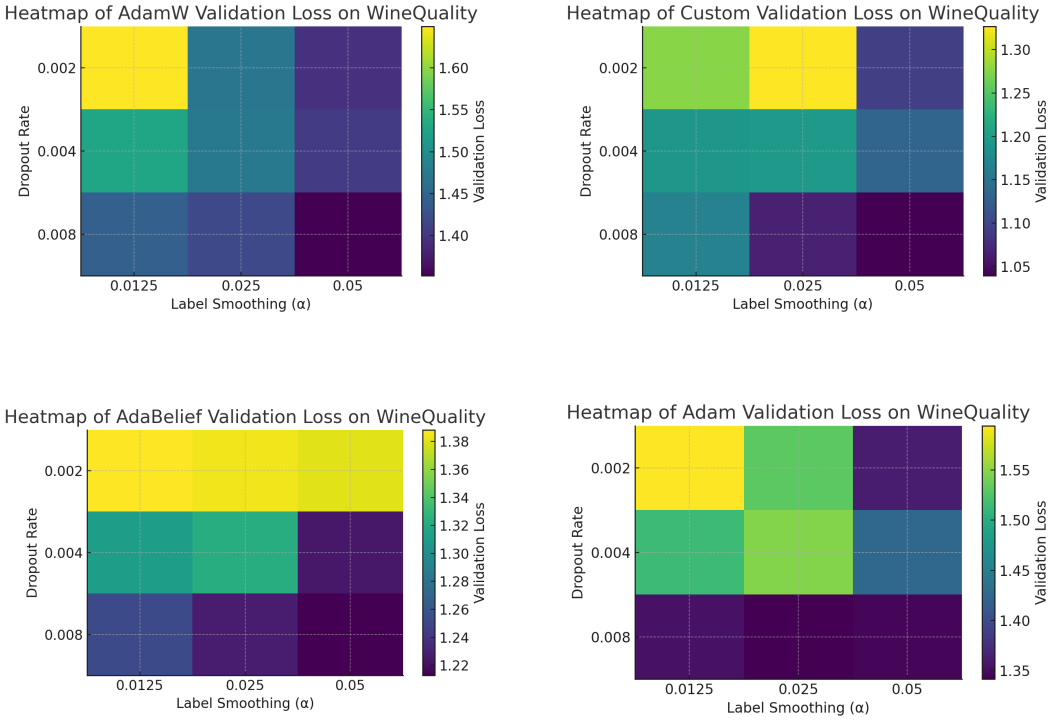


Figure 5: Heatmap of Hyperparameter Sensitivity Analysis