



Optimizers Performance is Task-Dependent: An Empirical Study of Learning Rate Sensitivity in Classification and Regression Tasks

Chisom ruth chibuike¹, Kelechi Edwin Onuigbo², Chika, Charles Ekene³

^{1,2,3}Department of Mathematics, University of Nigeria, Nsukka, Enugu State, Nigeria.

ARTICLE INFO	ABSTRACT
Published Online: 26 May 2026 Corresponding Author: Chika, Charles Ekene	The performance of deep neural networks is critically influenced by the choice of optimization algorithm and its hyperparameters, particularly the learning rate (η). However, the interplay between optimizers and learning rates across different task paradigms, such as classification and regression, remains under-explored. This paper addresses this gap by presenting a systematic empirical study of four common Optimizers (Adaptive Moment Estimation (Adam), Adaptive Gradient Algorithm (Adagrad), Root Mean Square Propagation(RMSProp), and Stochastic Gradient Descent(SGD) with Nesterov Accelerated Gradient momentum) across a spectrum of learning rates($\eta \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$). We evaluate 32 configurations on two distinct tasks: a Convolutional Neural Network (CNN) for image classification and a Multilayer Perceptron (MLP) for tabular regression. Our findings demonstrate that optimizer performance, stability, and learning rate sensitivity are highly task-dependent. We show that the optimal configuration diverges significantly between tasks: the CNN classification task achieved peak accuracy with RMSProp at a low $\eta = 10^{-4}$, whereas the MLP regression task achieved the highest R2 score (0.9102) with the non-adaptive SGD at a moderate $\eta = 10^{-2}$. These results provide strong empirical evidence that optimization strategies must be carefully tailored to the specific task and architecture.
KEYWORDS: Machine Learning, Deep Learning, Optimization, Learning Rates, Empirical Study, Convolutional Neural Networks, Multilayer Perceptron.	

I. INTRODUCTION

This template provides all the necessary information to the The performance and generalization ability of deep neural networks such as Convolutional Neural Networks (CNNs) and Multilayer Perceptrons (MLPs) are highly influenced by the optimization strategy employed during [1]. Central to this strategy is the choice of the optimization algorithm and its hyperparameters, particularly the learning rate (η). The learning rate determines the magnitude of parameter updates at each iteration of gradient descent and controls the trade-off between convergence speed and training stability. An inappropriate learning rate can cause divergence, oscillatory behavior, or convergence to suboptimal minima [2; 3].

While numerous optimizers have been developed to improve convergence and stability, their performance is inherently affected by their chosen learning rate on the considered task. Despite extensive usage across domains[4; 5], the interplay between learning rate and

optimizer behavior remains largely empirical. In particular, the effect of varying learning rates on optimizer performance across different task types and architectures has little systematic study. Most existing research often evaluates optimizers within a single context—either classification or regression—without examining how learning rate sensitivity may differ between these paradigms[4; 5; 6].

This study addresses this gap by conducting a comparative analysis of four widely used optimizers—Adaptive Moment Estimation (Adam)[7], Adaptive Gradient Algorithm (Adagrad)[8], Root Mean Square Propagation (RMSProp)[9], and Stochastic Gradient Descent (SGD) with Nesterov momentum[11; 12]—across a spectrum of learning rates ($\eta \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$). The evaluation spans two distinct deep learning architectures:

- Image Classification with CNNs: We implement a convolutional architecture for an appliance detection task, assessing how learning rate variations influence optimizers in a high-dimensional setting.
- Regression with MLPs: We train a Multi-Layer Perceptron for salary prediction on structured tabular data, examining how learning rate affects optimizer precision and error minimization in a continuous-output task.

We analyzed the effect of learning rates on performance across both architectures and task types, aiming to uncover the differential influence of learning rate on optimizer behavior in both classification and regression contexts. The principal contribution of this work is to provide empirical evidence and actionable insights on how learning rate choices modulate optimizer effectiveness across distinct deep learning tasks. Our empirical results reveal that optimizer performance and learning rate sensitivity are highly task-dependent. We demonstrate that the optimal optimization strategy diverges significantly between the high-dimensional image classification task and the tabular regression task, highlighting the need for task-specific tuning.

The rest of this paper is structured as follows: Section 2 reviews relevant literature on optimization algorithms and learning rate adaptation. Section 3 details the model architectures, datasets, and experimental setup. Section 4 presents and discusses the empirical findings. Finally, Section 5 concludes with key insights, limitations, and potential directions for future work.

II. LITERATURE REVIEW

In recent years, numerous studies have examined how the learning rate influences model convergence and generalization, emphasizing its central role across a wide range of machine learning applications. Although modern optimizers aim to reduce sensitivity to hyper-parameters, the choice of base learning rate remains fundamentally influential.

For instance, in the study "Demystifying Learning Rate Policies for High Accuracy Training of Deep Neural Networks"[3], the authors implemented their LRBench system to evaluate 13 learning-rate functions across various tasks and discovered that cyclic and decaying schedules improved convergence for non-adaptive optimizers, though adaptive optimizers (e.g., Adam) still required an appropriate base learning-rate to perform well. This emphasizes that learning-rate policies cannot be divorced from optimizer design nor from task context.

Similarly, the work by LLR: Learning Learning Rates by LSTM for Training Neural Networks adopts a meta-learning viewpoint: an LSTM controller generates per-step learning rates based on observed gradient/loss

patterns, leading to improved results for optimizers like SGD and Momentum. Interestingly, the authors note that adaptive optimizers such as RMSProp exhibited instability under their regime, pointing to how optimizer structure influences the interplay with learning-rate dynamics.

More recently, research has explored fully adaptive optimizers which incorporate dynamic learning-rate adjustment tied explicitly to loss behaviour. For example, the proposed AdaLo: Adaptive Learning Rate Optimizer with Loss for Classification[13] introduces an optimizer that increases step-size when the loss stagnates and reduces it when the loss plateaus, outperforming Adam and AdaBelief in convergence speed and stability. This highlights the potential of loss-aware learning-rate adaptation as a means to improve optimizer generalization across tasks.

In aggregate, these studies make clear that (i) the choice of learning-rate schedule and magnitude is deeply coupled with the optimizer type, (ii) learning-rate sensitivity may vary by task or architecture (classification vs. regression, image vs. tabular), and (iii) there is a growing movement toward optimizer designs that internalize learning-rate adaptation rather than treating learning rate as an external hyperparameter. Nonetheless, systematic investigation of how learning-rate sensitivity interacts with optimizer behaviour across different task types and architectures remains limited. This gap motivates our current study and contribution, which empirically compares four widely used optimizers over a discrete set of learning-rates across both a high-dimensional CNN classification task and a tabular MLP regression task—to reveal the task-dependent nature of learning-rate/optimizer interactions.

III. BENCHMARKING AND EXPERIMENTS

To rigorously evaluate the influence of learning rate (η) on optimizer performance across distinct learning paradigms, we implemented a structured benchmarking framework encompassing two representative tasks: tabular regression and image classification. This section details the datasets, preprocessing procedures, model architectures, and experimental design. To ensure reproducibility, all stochastic operations including data shuffling, weight initialization, and dataset partitioning were controlled by a fixed global random seed.

A. Datasets and Preprocessing

Two publicly available datasets were employed to represent heterogeneous learning scenarios. For the regression task, the Salary Prediction Dataset¹ was utilized. The dataset

¹. Available at:

<https://www.kaggle.com/datasets/rkiattisak/salary-prediction-for-beginner/data>

includes six variables: Age, Gender, Education Level, Job Title, Years of Experience, and Salary. Preprocessing involved the removal of the high-cardinality Job Title feature, elimination of rows containing missing values, and one-hot encoding of categorical variables (Gender and Education Level). We also isolated the target variable, Salary, for prediction. Data were deterministically partitioned into 80% training and 20% testing subsets, with 20% of the training data partition, used for validation, yielding an effective 64%/16% /20% train/validation/test split overall. All numerical features and the target variable were standardized to zero mean and unit variance using StandardScaler and fitted exclusively on the training subset.

For the classification task, the Mechanical Tools Classification Dataset²[14] was used. Here, our experiments focused on the train_data_V2 subset containing three classes— hammer, screwdriver, and wrench. For preprocessing, all images were resized to 300 x 300 pixels and rescaled from the [0,255] to the [0,1] range. To enhance generalization, extensive data augmentation was applied during training, including random rotations (up to 30°), horizontal and vertical flips, width and height shifts (≤20%), shear and zoom transformations (≤20%), and brightness variations within ±30%.

B. Model Architectures

We designed distinct neural architectures to reflect the nature of each task, with careful consideration given to the mathematical properties of the chosen components.

Multilayer Perceptron (MLP) for Regression: For the regression task, we implemented a multilayer perceptron (MLP). The architecture comprised an input layer corresponding to the number of preprocessed features, followed by four hidden layers with 128, 64, 32, and 16 neurons, respectively. Each of the first three layers employed the Rectified Linear Unit (ReLU) activation, defined as $f(z) = \max(0, z)$, which was chosen to introduce non-linearity while mitigating the vanishing gradient problem common in activation function. These layers also included batch normalization and dropout regularization (rates 0.3, 0.3, and 0.2). The output layer contained a single neuron with a linear activation function, $f(z) = z$, as it allows the model to predict an unbounded, real-valued number (the standardized salary) without constraining the output range.

Convolutional Neural Network (CNN) for Classification: For the image classification task, we developed a convolutional neural network (CNN). The

model accepts $300 \times 300 \times 3$ input tensors and consisted of successive convolutional blocks with filter counts of 64, 128, 256, and 512. Each block integrated convolutional layers with 3×3 kernels, batch normalization, and max pooling. The deeper blocks employed two stacked convolutional layers before pooling. All convolutional and dense layers in the CNN used Leaky ReLU activations, $f(z) = \max(\alpha z, z)$ with a small slope $\alpha > 0$, to address the "Dying ReLU" problem and prevent neurons from becoming permanently inactive during the long training cycles typical of image tasks. L2 regularization ($\lambda = 0.001$) was applied to the kernel weights to control complexity. We followed the convolutional backbone with a classification head comprising a global average pooling layer, two fully connected layers (512 and 256 units), and dropout ($p = 0.4$). The final layer applied the softmax activation function which is given as:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}, \text{ for } i = 1, \dots, K \quad (1)$$

where K is the number of classes, to convert the network's raw output scores (z) into a probability distribution over the tool categories.

C. Experiments Configuration

We used a grid-based benchmarking strategy to evaluate the interaction between optimizers and learning rates (η). We selected four widely-used optimizers: Adam, Adagrad, RMSProp, and Stochastic Gradient Descent (SGD) with Nesterov momentum ($\mu = 0.9$). We tested each optimizer with four fixed learning rates: $\eta \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$. This range was selected based on typical recommended settings across deep learning applications. This yielded 16 optimizer–learning rate configurations per task and a total of 32 independent experimental runs. The general parameter update rule for an iterative optimization process, where the update is scaled by the learning rate (η), is given by:

$$w_{t+1} = w_t - \eta \cdot g(w_t, \text{history}) \quad (2)$$

where w_t is the weight vector at time step t, η is the learning rate, and g is the calculated update direction incorporating the gradient and, potentially, historical information. This varies significantly between optimizers (e.g., standard gradient for SGD, with adaptive scaling for Adam, Adagrad, and RMSProp). The detailed mathematical equations for each optimizer are provided in Appendix A. Table I summarizes the primary task-specific training parameters. All models were trained under identical conditions to ensure training stability.

Table I: Task-specific training hyperparameters for both regression and classification tasks.

Hyperparameter	MLP (Regression)	CNN (Classification)
----------------	---------------------	-------------------------

². Available at:

<https://www.kaggle.com/datasets/salmaneunus/mechanical-tools-dataset/data>

Epochs	100	50
Batch size	32	64
Loss function	Mean Squared Error	Sparse Categorical Crossentropy

D. Evaluation

We evaluated model performance using metrics tailored to each learning paradigm. For the regression task, we assessed the performance on the test-set using three metrics:

- **Root Mean Square Error (RMSE):** This is calculated as $RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \gamma_i)^2}$. This measures the standard deviation of the residuals (prediction errors) and quantifies the model's typical error magnitude in the same units as the target variable, while placing greater weight on larger errors.
- **Mean Absolute Error (MAE):** Calculated as $MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \gamma_i|$. MAE represents the average magnitude of errors.
- **Coefficient of Determination (R2):** Calculated as $R^2 = 1 - \frac{\sum_i (y_i - \gamma_i)^2}{\sum_i (y_i - \bar{Y}_i)^2}$. This metric quantifies the proportion of the variance in the dependent variable explained by the independent variables.

In these equations, y_i represents the actual observed value for the i^{th} observation, γ_i denotes the corresponding value predicted by the model, and $\bar{Y}_i$ represents the arithmetic mean of all actual observed values. For the classification task, we monitored training and validation accuracies. The final validation accuracy served as the primary comparison criterion within the 50-epoch training schedule, representing the model's ability to generalize to unseen data.

We aggregated the results from all 32 experimental configurations to facilitate a systematic comparison of how learning rate variations influence optimizer behavior across both tasks. The empirical findings and their implications are presented in Section 4.

IV. EMPIRICAL FINDINGS AND RESULTS

A. Datasets and Preprocessing

Our CNN-based appliance classification task revealed substantial sensitivity to learning rate selection across all optimizers. Table 2 presents a comprehensive summary of the experimental outcomes, while Figures 1, 2, and 3 illustrate the training dynamics and final performance for each optimizer–learning rate combination.

• Learning Rate Effects Across Optimizers

Extremely Small Learning Rate ($\eta = 10^{-4}$): At the lowest learning rate, adaptive optimizers demonstrated superior performance. RMSProp achieved the highest validation accuracy of 58.33% (epoch 46), closely followed by Adam at 56.92% (epoch 46), as shown in Table 2 and Figure 3. Both optimizers exhibited smooth, monotonic convergence with minimal oscillation in validation loss (Figure I). We also noticed overfitting in Adagrad with this learning rate.

Small Learning Rate ($\eta = 10^{-3}$): Performance degraded across most optimizers at this intermediate learning rate.

SGD was more robust to this learning rate, reaching a 53.25% (epoch 35) and showing competitive performance. The loss curves (Figure I) shown reveals increased validation loss oscillations for Adam and RMSProp, particularly in later epochs, indicating reduced stability which suggests overshooting around local minima.

Moderate Learning Rate ($\eta = 10^{-2}$): At this learning rate, performance varied significantly. Adagrad achieved its highest validation accuracy at 55.33%, demonstrating remarkable robustness to increases in the learning rate. SGD also followed a similar pattern, achieving 54.08%. In contrast, Adam and RMSProp both plateaued at lower accuracies respectively. The training curves showed rapid initial descent followed by erratic validation loss behavior, (Figure I).

Large Learning Rate ($\eta = 10^{-1}$): The highest learning rate induced catastrophic training failures across multiple optimizers. RMSProp experienced complete divergence, with validation loss spiking into the hundreds of thousands (Figure I, bottom right). It achieved a peak validation accuracy of 39.25% before this instability, but its final accuracy was only 33.75% (Figure II). Adam and SGD showed marginally better stability, though with highly volatile training dynamics. Adagrad demonstrated the most resilience at this extreme learning rate, reaching 53.75%, likely due to its adaptive per-parameter learning rate scaling that inherently dampens aggressive updates.

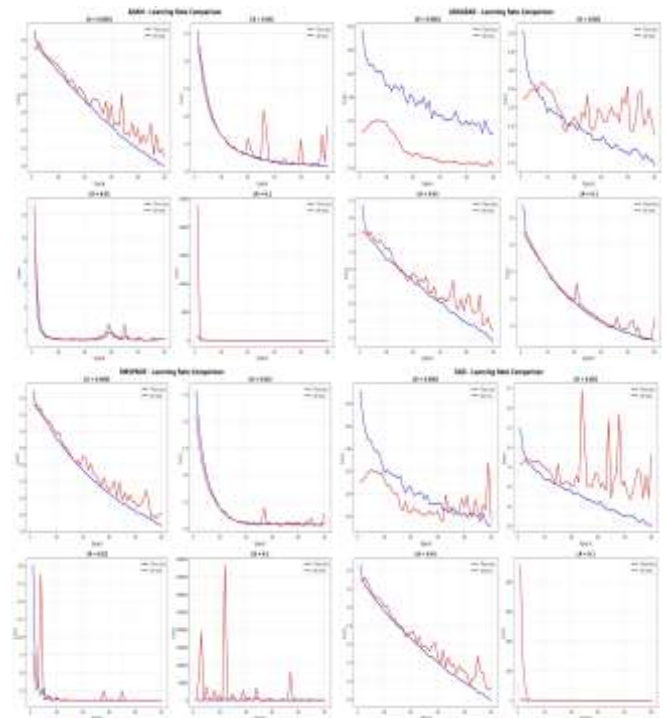


Figure I: Training and validation loss curves for the CNN classification task (Image Prediction) across all four optimizers (Adam, Adagrad, RMSProp, SGD) and four learning rates (10-4, 10-3, 10-2, 10-1).

Table II: Summary of CNN classification performance for Image Prediction. Our best validation accuracy and the epoch it was achieved.

Optimizer	Learning Rate	Best Validation Acc. (%)	Best Epoch
ADAM	0.0001	56.92%	46
ADAM	0.001	48.08%	38
ADAM	0.01	46.58%	16
ADAM	0.1	44.00%	38
ADAGRAD	0.0001	50.25%	38
ADAGRAD	0.001	52.83%	42
ADAGRAD	0.01	55.33%	50
ADAGRAD	0.1	53.75%	45
RMSPROP	0.0001	58.33%	46
RMSPROP	0.001	50.58%	42
RMSPROP	0.01	43.33%	26
RMSPROP	0.1	39.25%	57*
SGD	0.0001	51.33%	46
SGD	0.001	53.25%	35
SGD	0.01	54.08%	42
SGD	0.1	52.92%	48

*Epoch 57 is beyond the 50-epoch training, indicating peak was likely at epoch 50.

- Optimizer-Specific Observations**

RMSProp: RMSProp demonstrated the highest peak performance (58.33% at $\eta = 10^{-4}$) but exhibited sensitivity to learning rate increases. Performance collapsed dramatically at $\eta = 10^{-2}$ and completely diverged at $\eta = 10^{-1}$, as evidenced by loss values in the hundreds of thousands (Figure I). The optimizer’s exponentially weighted moving average of squared gradients appears to provide insufficient damping at higher learning rates for this classification task.

Adam: Adam showed consistent but suboptimal performance across all learning rates. As seen in Figure 3, it never achieved the highest accuracy at any setting. It exhibits no overfitting. However, it’s loss curves also revealed characteristic "spiky" validation behavior at higher learning rates, suggesting difficulty in fine-tuning near convergence.

Adagrad: Adagrad emerged as the most robust optimizer, achieving competitive performance across the entire learning rate spectrum (Figure III). Its monotonically decreasing learning rate schedule prevented the aggressive updates that destabilized other optimizers at high learning rates. Notably, Adagrad achieved its best performance at the moderate learning rate of $\eta = 10^{-2}$ (55.33%), but exhibited overfitting at inital smaller learning rates.

SGD with Nesterov Momentum: SGD with Nesterov momentum displayed moderate and consistent performance across all learning rates, with best validation accuracies

ranging from 51.33% to 54.08%. SGD also exhibited inconsistency in training stability for this classification task.

- Convergence Dynamics and Training Stability**

Examination of the loss curves (Figure 1) revealed distinct convergence patterns. At $\eta = 10^{-4}$, all optimizers demonstrated smooth exponential decay with minimal divergence between training and validation losses. As learning rates increased, a consistent pattern emerged: rapid initial descent followed by increasing validation loss volatility. This behavior was most pronounced in RMSProp and Adam at $\eta \geq 10^{-2}$, manifesting as sharp spikes in validation loss despite continued training loss reduction, a clear indicator of poor stability.

The final validation accuracy heatmap (Figure II) illustrates the inverse relationship between learning rate and final performance for Adam and RMSProp, with a sharp performance cliff at $\eta = 10^{-1}$. In contrast, Adagrad and SGD exhibited more gradual performance degradation, suggesting greater robustness to hyperparameter misspecification.

 - Best Configuration**

The optimal configuration for this CNN classification task with our mentioned architecture was RMSProp with $\eta = 10^{-4}$, achieving 58.33% validation accuracy at epoch 46 (Table II). However, considering robustness to learning rate selection, Adagrad at $\eta = 10^{-2}$ (55.33%, epoch 50) presents a more practical choice, offering strong performance with reduced sensitivity to hyperparameter tuning. This results emphasize the importance of learning rate in optimizers, where even moderate deviations from optimal values can substantially degrade performance or induce training instability.

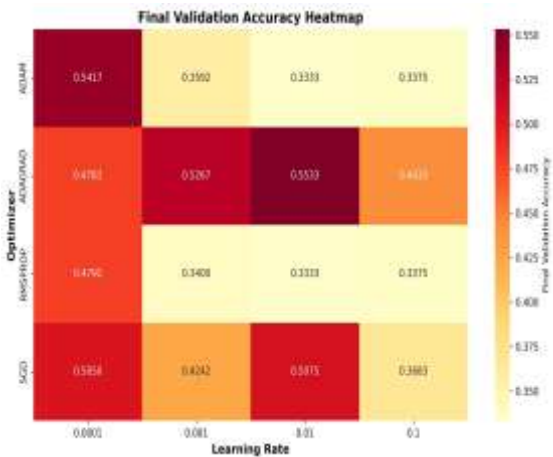


Figure II: Heatmap of final validation accuracy. This shows the accuracy at the end of the 50-epoch training, highlighting stability and final convergence for the

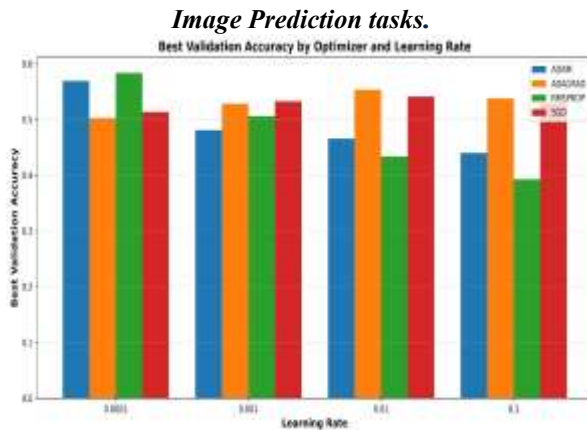


Figure III: Bar chart of the best validation accuracy achieved by each optimizer and learning rate combination at any point during training.

B. Salary Prediction with MLP

In our MLP-based salary prediction task, the choice of optimizer and learning rate had a profound impact on model convergence and predictive accuracy. We evaluated test performance using the root mean squared error (RMSE), mean absolute error (MAE), and coefficient of determination (R2). The comprehensive test set results are detailed in Table III, with training dynamics illustrated in Figure IV. A summary of test RMSE and R2 performance is shown in Figure V, and the best predictive models for each optimizer are visualized in Figure VI.

• Learning Rate Effects Across Optimizers

Extremely Small Learning Rate ($\eta = 10^{-4}$): At this low learning rate, Adagrad failed to learn entirely, resulting in a negative R2 score (-0.0658) and an extremely high RMSE (50,551), as seen in Table III. The loss curves (Figure 4) confirm its validation loss stagnated. Adam, RMSProp, and SGD performed comparably, achieving respectable R2 scores between 0.81 and 0.83. However, their loss curves all show a large, persistent gap between training and validation loss, indicative of underfitting.

Small Learning Rate ($\eta = 10^{-3}$): This learning rate proved effective for adaptive optimizers for this task. RMSProp achieved its best performance here, with an R2 of 0.8983 and a low RMSE of 15,618. Adam and Adagrad also performed better. This represents a clear improvement for all optimizers over a learning rate of 10^{-4} .

Moderate Learning Rate ($\eta = 10^{-2}$): This learning rate yielded the best overall results. SGD achieved the top performance of all 16 configurations, with a test R2 of 0.9102 and the lowest RMSE. Adam also achieved its peak performance at this rate (R2 = 0.9003). In contrast, RMSProp's performance began to degrade, and its loss curve (Figure IV) shows increased validation loss volatility, suggesting the onset of instability.

Large Learning Rate ($\eta = 10^{-1}$): The highest learning rate caused RMSProp to diverge completely, resulting in a negative R2 score (-3.0266) and an RMSE over 98,000. This is visualized as a "hot" cell in the heatmap (Figure V)

and confirmed by the explosive validation loss in its plot (Figure IV). SGD's performance also dropped significantly. Adam remained surprisingly stable, maintaining a high R2 of 0.8895. Notably, Adagrad achieved its best performance at this high learning rate, demonstrating a sensitivity profile distinct from the other optimizers.

• Optimizer-Specific Observations

Adam: Adam demonstrated high robustness, achieving strong performance at all learning rates except the lowest. It achieved its peak performance ($\eta = 10^{-2}$) with an R2 of 0.9003 and RMSE of 15,463.

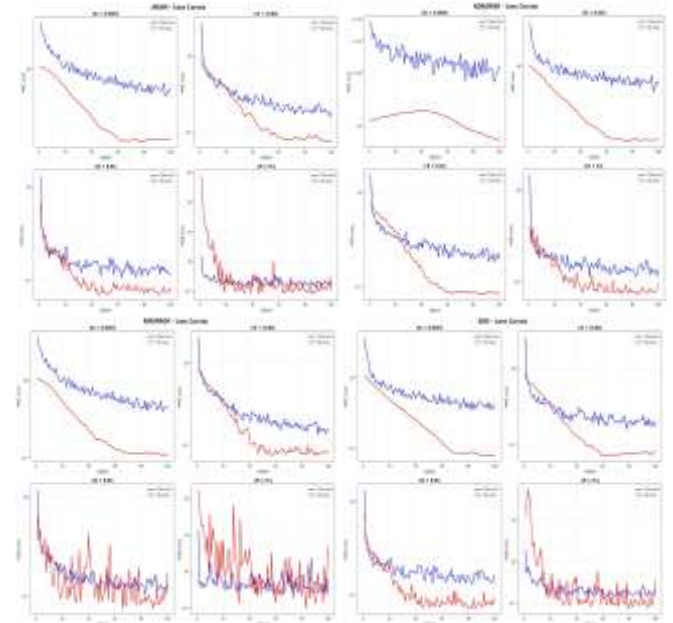


Figure IV: Training and validation loss curves for the MLP architecture (salary prediction task) across all four optimizers and four learning rates.

Adagrad: Adagrad's performance was inversely correlated with its behavior in the CNN task. It performed extremely poorly at low learning rates and, counter-intuitively, achieved its best result at the highest learning rate, $\eta = 10^{-1}$. This suggests the tabular task benefited from Adagrad's aggressive initial steps, which were only possible with a large learning rate.

RMSProp: RMSProp was the least stable optimizer for this task. While it achieved an excellent peak performance at $\eta = 10^{-3}$, it was highly sensitive to learning rate. Its performance degraded at 10^{-2} and it experienced catastrophic divergence at 10^{-1} , as shown clearly in Figure 5.

SGD with Nesterov Momentum: SGD exhibited a "classic" hyperparameter tuning profile: it underfit at $\eta = 10^{-4}$, improved at $\eta = 10^{-3}$, found an optimal "sweet spot" at $\eta = 10^{-2}$, and then overshot, leading to degraded performance at $\eta = 10^{-1}$. Its optimal configuration produced the best results of any optimizer, as seen in its

scatter plot (Figure VI), which shows the tightest clustering around the perfect-prediction line.

• **Convergence Dynamics and Predictive Quality**

The training dynamics for this regression task (Figure IV) were notably different from the classification task. For most successful runs (e.g., Adam at 10^{-2} , SGD at 10^{-2}), the validation loss (red) consistently trended below the training loss (blue). This is a known artifact of using dropout, which is active during training but disabled during validation, effectively making the validation network simpler and less noisy. This indicates that our regularization scheme was effective and did not lead to overfitting.

Our predictive quality, visualized in Figure VI, is high for the best-performing models. All four optimizers, when tuned to their best learning rate, produced models with R2 scores near 0.9, indicating they could explain approximately 90% of the variance in salary. The "failed" configurations are clearly visible in the R2 comparison chart (Figure V), which highlights the negative R2 scores for Adagrad ($\eta = 10^{-4}$) and RMSProp ($\eta = 10^{-1}$), signifying models that performed worse than a simple mean-value baseline.

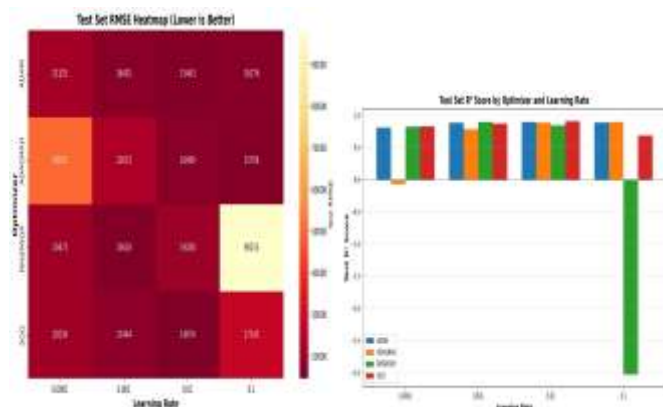


Figure V: Test set performance metrics for the MLP architecture (salary prediction task). On the left is a heatmap of our Test RMSE, where lighter colors indicate worse performance. On the right is our R2 test Score, where values closer to 1.0 are better.

Table III: Summary of MLP regression performance (salary prediction task) on the test set for our salary prediction task.

Optimizer	Learning Rate	Test RMSE	Test MAE	Test R2
ADAM	0.0001	21,325	15,590	0.8103
ADAM	0.001	16,401	10,571	0.8878
ADAM	0.01	15,463	9,963	0.9003
ADAM	0.1	16,276	10,746	0.8895
ADAG	0.0001	50,551	40,812	-0.0658

RAD				
ADAG	0.001	22,633	16,084	0.7864
RAD				
ADAG	0.01	16,489	11,394	0.8866
RAD				
ADAG	0.1	15,706	10,581	0.8971
RAD				
RMSPR	0.0001	20,473	14,552	0.8252
OP				
RMSPR	0.001	15,618	9,606	0.8983
OP				
RMSPR	0.01	19,280	13,384	0.8450
OP				
RMSPR	0.1	98,255	64,097	-3.0266
OP				
SGD	0.0001	20,234	12,806	0.8292
SGD	0.001	17,444	10,724	0.8731
SGD	0.01	14,674	9,370	0.9102
SGD	0.1	27,343	22,076	0.6882

• **Best Configuration**

The optimal configuration for the MLP regression task was SGD with $\eta = 10^{-2}$. This combination achieved the highest test R2 score (0.9102) and the lowest test RMSE (14,674) and MAE (9,370), as detailed in Table III. Close contenders were Adam at $\eta = 10^{-2}$ and RMSProp at $\eta = 10^{-3}$. The results for this task suggest that for this MLP architecture, the classic SGD with Nesterov momentum, when its learning rate is well-tuned, can outperform adaptive optimizers.

V. CONCLUSION

This study conducted a systematic empirical analysis of the interplay between four common optimizers (Adam, Adagrad, RMSProp, and SGD) and a spectrum of learning rates ($\eta \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$) across two distinct deep learning architectures: CNN-based image classification and MLP-based tabular regression. Our results compellingly demonstrate that optimizer performance and learning rate sensitivity are highly task-dependent, challenging any "one-size-fits-all" approach to optimization.

The principal contribution of this work is the empirical evidence from direct-comparison showing that optimal optimization strategies diverge significantly between these two tasks. For the high-dimensional CNN classification task, the optimal configuration was RMSProp at a low $\eta = 10^{-4}$, which achieved the highest accuracy (58.33%). However, this optimizer was extremely sensitive, showing catastrophic divergence at η

= 10⁻¹. In stark contrast, the MLP regression task favored a higher learning rate "sweet spot," with the non-adaptive SGD at a moderate $\eta = 10^{-2}$ achieving the best-in-class performance (R² = 0.9102).

These results suggest that the optimization landscape of a deep CNN benefits from the small, careful steps of a low- η adaptive optimizer or the high intrinsic stability of Adagrad. In contrast, the MLP on structured data benefited from the larger, momentum-driven steps of SGD or the robustness of Adam at a higher learning rate.

Limitations and Future Work. Our findings are bound by its experimental scope. We focused on two specific datasets and model architectures, and a discrete set of four learning rates. We did not investigate other hyperparameters (e.g., momentum, β_1, β_2) or the effect of dynamic learning rate schedules (e.g., decay, warmup), which are common practice and could significantly alter optimizer stability and final performance.

Future work should expand this benchmark to include a wider array of architectures (e.g., Transformers, GNNs) and data modalities (e.g., text, time-series). Further investigation into the interaction between learning rate schedules and optimizer choice would provide deeper insights. Ultimately, our work provides actionable evidence that the optimal optimization strategy is not universal, but is instead a nuanced choice that must be tailored to the specific architecture and task at hand.

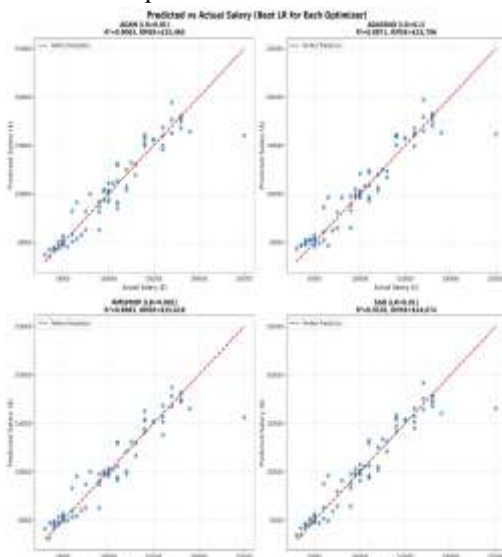


Figure VI: Predicted vs. Actual Salary for the best-performing configuration of each optimizer on the test set. The dashed red line represents a perfect prediction.

REFERENCES

1. Z. Lu, Y. Sun, Z. Yang, Y. Zhang, P. Tulajiang, H. Sun, and H. Lin, "Improving generalization in DNNs through enhanced orthogonality in momentum-based optimizers," *Inf. Process. Manag.*, vol. 62, no. 4, Art. no. 104109, 2025, doi: 10.1016/j.ipm.2025.104109.

2. A. Atamna, T. Maus, F. Kievelitz, and T. Glasmachers, "Cumulative learning rate adaptation: Revisiting path-based schedules for SGD and Adam," *arXiv:2508.05408*, 2025. [Online]. Available: <https://arxiv.org/abs/2508.05408>.
3. Y. Wu, L. Liu, J. Bae, K.-H. Chow, A. Iyengar, C. Pu, W. Wei, L. Yu, and Q. Zhang, "Demystifying learning rate policies for high accuracy training of deep neural networks," *arXiv:1908.06477*, 2019. [Online]. Available: <https://arxiv.org/abs/1908.06477>.
4. C. E. Chika, "Machine learning approach and model for predicting proton stopping power ratio and other parameters using computed tomography images," *J. Med. Phys.*, vol. 49, no. 4, pp. 519–530, 2024, doi: 10.4103/jmp.jmp_120_24.
5. C. E. Chika, "Estimation of proton stopping power ratio and mean excitation energy using electron density and its applications via machine learning approach," *J. Med. Phys.*, vol. 49, no. 2, pp. 155–166, 2024, doi: 10.4103/jmp.jmp_157_23.
6. R. Zaheer and H. Shaziya, "A Study of the Optimization Algorithms in Deep Learning," 2019 Third International Conference on Inventive Systems and Control (ICISC), Coimbatore, India, pp. 536–539, 2019, doi: 10.1109/ICISC44355.2019.9036442.
7. C. Peng, "Comprehensive analysis of the impact of learning rate and dropout rate on the performance of convolutional neural networks on the CIFAR-10 dataset," *Appl. Comput. Eng.*, vol. 102, pp. 183–192, 2024, doi: 10.54254/2755-2721/102/20241161.
8. I. A. Dewi and M. A. N. E. Salawangi, "High performance of optimizers in deep learning for cloth patterns detection," *IAES Int. J. Artif. Intell.*, vol. 12, no. 3, pp. 1407–1418, 2023, doi: 10.11591/ijai.v12.i3.
9. D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv:1412.6980*, 2017. [Online]. Available: <https://arxiv.org/abs/1412.6980>.
10. J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, no. 61, pp. 2121–2159, 2011.
11. T. Tieleman and G. Hinton, "Lecture 6.5—RMSProp: Divide the learning rate by a running average of its recent magnitude," *Neural Netw. Mach. Learn.*, Coursera Lecture Notes, 2012.
12. Y. E. Nesterov, "A method for solving the convex programming problem with convergence rate $O(1/k^2)$," *Sov. Math. Dokl.*, vol. 27, pp. 372–376, 1983.
13. C. Yu, X. Qi, H. Ma, X. He, C. Wang, and Y. Zhao, "LLR: Learning learning rates by LSTM for training neural networks," *Neurocomputing*, vol. 394, pp. 41–50, 2020, doi: 10.1016/j.neucom.2020.01.106.
14. J. J. Jeong and G. Koo, "AdaLo: Adaptive learning rate optimizer with loss for classification," *Inf. Sci.*, vol.

690, Art. no. 121607, 2025, doi: 10.1016/j.ins.2024.121607.

15. S. Hossain, J. Komol, and M. M. Raidah, “Mechanical tools classification dataset,” Kaggle Dataset, 2020.

Appendix A. Optimizer Algorithms

This appendix provides the formal mathematical definitions for the four optimizers—Stochastic Gradient Descent (SGD) with Nesterov Momentum, Adagrad, RMSProp, and Adam—that were benchmarked in our experiments.

A.1 Stochastic Gradient Descent (SGD) with Nesterov Momentum

SGD with Nesterov Momentum incorporates a look-ahead mechanism. The velocity v_t is calculated by evaluating the gradient g_t at the momentum-adjusted position $w_t + \mu v_t - 1$, where μ is the momentum coefficient.

Algorithm 1 SGD with Nesterov Momentum

Require: Learning rate η , momentum coefficient $\mu = 0.9$

- 1: Initialize: $v_0 = 0$
- 2: for $t = 0, 1, 2, \dots$ do
- 3: $\tilde{w}_t \leftarrow w_t + \mu v_t - 1$ {Look-ahead step}
- 4: $g_t \leftarrow \nabla L(\tilde{w}_t)$ {Gradient at look-ahead point}
- 5: $v_t \leftarrow \mu v_t - 1 + \eta g_t$ {Velocity update}
- 6: $w_{t+1} \leftarrow w_t - v_t$ {Parameter update}
- 7: end for

A.2 Adagrad (Adaptive Gradient)

Adagrad scales the learning rate for each parameter inversely proportional to the square root of the sum of all its historical squared gradients, thereby providing larger updates for infrequent parameters.

Algorithm 2 Adagrad

Require: Learning rate η , smoothing term $\epsilon = 10^{-8}$

- 1: Initialize: $s_0 = 0$
- 2: for $t = 0, 1, 2, \dots$ do
- 3: $g_t \leftarrow \nabla L(w_t)$
- 4: $S_t \leftarrow S_{t-1} + g_t \odot g_t$ {Accumulate squared gradients, where $\odot$ denotes element-wise (Hadamard) multiplication}
- 5: $w_{t+1} \leftarrow w_t - \frac{\eta}{\sqrt{S_t + \epsilon}} \odot g_t$
- 6: end for

A.3 RMSProp (Root Mean Square Propagation)

RMSProp uses an exponentially decaying average of squared gradients, governed by the decay rate γ , to resolve Adagrad’s aggressive decay of the learning rate.

Algorithm 3 RMSProp

Require: Learning rate η , decay rate $\gamma = 0.9$, smoothing term $\epsilon = 10^{-8}$

- 1: Initialize: $s_0 = 0$
- 2: for $t = 0, 1, 2, \dots$ do
- 3: $g_t \leftarrow \nabla L(w_t)$
- 4: $s_t \leftarrow \gamma s_{t-1} + (1 - \gamma) g_t \odot g_t$ {Decaying average}
- 5: $w_{t+1} \leftarrow w_t - \frac{\eta}{\sqrt{s_t + \epsilon}} \odot g_t$
- 6: end for

A.4 Adam (Adaptive Moment Estimation)

Adam combines the momentum-like properties of the first moment estimate (m_t) with the adaptive learning rates of the second moment estimate (s_t), along with bias-correction steps for both moments.

Algorithm 4 Adam

Require: Learning rate η , decay rates $\beta_1 = 0.9$, $\beta_2 = 0.999$, smoothing term $\epsilon = 10^{-8}$

- 1: Initialize: $m_0 = 0$, $s_0 = 0$, $t = 0$
- 2: while stopping criterion not met do
- 3: $t \leftarrow t + 1$
- 4: $g_t \leftarrow \nabla L(w_t)$
- 5: $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$ {First moment estimate}
- 6: $s_t \leftarrow \beta_2 s_{t-1} + (1 - \beta_2) g_t \odot g_t$ {Second moment estimate}
- 7: $\tilde{m}_t \leftarrow \frac{m_t}{1 - \beta_1^t}$ {Bias correction for first moment}
- 8: $\tilde{s}_t \leftarrow \frac{s_t}{1 - \beta_2^t}$ {Bias correction for second moment}
- 9: $w_{t+1} \leftarrow w_t - \frac{\eta}{\sqrt{\tilde{s}_t + \epsilon}} \odot \tilde{m}_t$
- 10: end while
