Proceedings of the 4th International Conference on Mechatronics and Smart Systems
DOI: 10.54254/2755-2721/2026.33717

Performance Advantage Comparison of Five Mainstream Optimizers on Datasets with Different Characteristics

Ziyi Li

School of Electonics and Information Engineering, Anhui Jianzhu University, Hefei, China
PflaumchenKami123@outlook.com

Abstract. This paper conducts two comparative experiments. One uses the Software Developer Salary Prediction Dataset, whose target variable is continuous and has a wide value range. The other uses the Red Wine Quality Dataset, whose target variable is quasi-discrete with a narrow value range and whose features are sparse. Five mainstream optimizers are selected for these two experiments: Stochastic Gradient Descent (SGD), Momentum, Adaptive Gradient Algorithm (AdaGrad), Root Mean Square Propagation (RMSProp), and Adaptive Moment Estimation (Adam). To ensure fair experimental implementation, parameters are unified in the experiments. The same weight initialization method is adopted for all optimizers. For datasets, min-max normalization is applied to salary data and Z-score normalization to wine quality data. In addition, a fixed random seed (42) is used throughout. No learning rate scheduling or early stopping is applied during the training of all optimizers, so that their convergence properties can be fully observed. Experimental results are recorded with high precision: two decimal places for salary prediction results and five decimal places for wine quality prediction results to ensure optimal visualization. Experimental results show that adaptive optimizers outperform basic optimizers, but the choice of the best optimizer still depends on the specific task. Finally, guidance is provided for dataset types in practical applications.

Keywords: Regression task, performance comparison, adaptive learning rate

1. Introduction

Lately machine learning has begun with the earliest optimizer batch gradient descent (BGD). It is an old and simple iterative algorithm that updates the model parameters with all the training data.It has a few drawbacks such as being slow in processing large data sets, consuming a lot of memory. In order to solve these problems, researchers used only the small part of the data in each iteration. This eventually led to stochastic gradient descent (SGD), which is faster and has larger memory usage [1]. One advantage of SGD is the random sampling feature, which may not have the best local solutions. As a consequence, SGD has become an important optimization tool in artificial intelligence. As time progressed people began to learn with SGD, slow convergence speed and large fluctuations in training. Momentum is one of the first classical optimization techniques for SGD [2]. It adds a "momentum" term when updates are conducted to help it to update in a stable direction and to reduce oscillations. In the late 2010s, adaptive learning rate methods emerged, including

DOI: 10.54254/2755-2721/2026.33717

AdaGrad, RMSprop, and Adam [3-5]. Although there are so many optimizers and their performance has improved when dealing with large datasets. The performance of various optimizers varies on different datasets. For example, the same optimizer produces completely different results on different datasets. Therefore, it is important for people to choose the appropriate optimizer for a specific task.

This paper focuses on the performance of the mainstream optimization algorithms on datasets with different features. And it provides targeted guidance for the optimization algorithms. It conducts comparative experiments on standard SGD, momentum method, AdaGrad, RMSProp, and Adam to analyze the advantages of each method and their applicable scenarios.

2. Method

2.1. Dataset preparation

2.1.1. Programmer salary prediction dataset

The programmer salary prediction dataset is a regression dataset. It models the relationship between employees' characteristics and their annual salary. This simulates real-world human resources salary prediction scenarios [6]. The dataset contains 6 core input features and 1 continuous target variable. It has 10,000 labeled samples. The samples are split into a training set and a test set at an 8:2 ratio (8,000 training samples and 2,000 test samples). This split ensures the model's generalization ability.

To ensure fair optimizer comparison, standardized preprocessing is applied to all features. Continuous features (e.g., age, work experience) are scaled to the [0, 1] range using min-max normalization. Ordinal features are kept as whole numbers. Categorical features are turned into one-hot encoded vectors to remove ordinal bias. A fixed random seed (42) is used for stratified splitting. This keeps the distribution of company size and job level the same in both training and test sets, so there is no distribution shift. The salary target is kept as original USD values (no scaling). This makes prediction results easy to understand. MSE loss is calculated directly on original values to show real prediction errors.

2.1.2. Red wine quality prediction dataset

The red wine quality prediction experiment uses the public UCI Red Wine Quality Dataset [7]. This is a standard benchmark for regression tasks in wine quality assessment. The dataset contains 1,599 samples of Portuguese red wine. It has 11 physicochemical input features and 1 sensory quality score target. The data are split into an 8:2 training-test set (1,279 training samples, 320 test samples) following the original dataset's standard split protocol.

All input features are standardized with Z-score normalization. This keeps the experiment consistent with the salary prediction task. It also ensures fair optimizer comparison. After standardization, each feature has a mean of 0 and a variance of 1. Scale differences between physicochemical indicators are removed. The gradient size of each feature also becomes the same during model training. A fixed random seed (42) is used for data splitting. This makes the experiment results repeatable. The test set is saved for high-precision prediction evaluation. The precision goes up to 5 decimal places. The quality score target is kept in its original range (3–8). MSE loss is calculated directly on the continuous prediction values. This gives a direct measure of regression prediction accuracy.

DOI: 10.54254/2755-2721/2026.33717

2.2. Optimization strategies

2.2.1. Stochastic Gradient Descent (SGD)

SGD is the foundational gradient-based optimization algorithm [8]. It updates model parameters by computing gradients on a single mini-batch of data, rather than on the full training set. This reduces computational costs. The core update rule is:

$$\theta_{t+1} = \theta_t - \eta \bullet g_t \tag{1}$$

where θ_t is the parameter at step t , η is the fixed learning rate, and $g_t = \nabla L(\theta_t)$ is the mini-batch gradient of the loss function. SGD uses a fixed learning rate, no momentum, and no adaptive gradient scaling. It converges slowly on high-dimensional or non-convex loss landscapes. It is prone to oscillation around local minima. However, it has low computational overhead and strong generalization on simple tasks. In the experiment, the hyperparameters are set as: initial learning rate $\eta = 0.001$, weight decay=0, momentum=0, Nesterov=False, representing pure SGD.

2.2.2. SGD with momentum

Momentum SGD enhances vanilla SGD by adding a momentum term. This term accelerates convergence and suppresses gradient oscillation. It is inspired by the physical concept of inertia. The core update rules are:

$$v_{t+1} = \beta \cdot v_t + \eta \cdot g_t \tag{2}$$

$$\theta_{t+1} = \theta_t - v_{t+1} \tag{3}$$

where v_t is the velocity term (accumulated historical gradient), and β is the momentum coefficient (default=0.9). Momentum SGD accelerates convergence in consistent gradient directions. It suppresses oscillation when gradient directions fluctuate. It outperforms vanilla SGD in most regression tasks. The hyperparameters are set as initial learning rate $\eta = 0.001$, momentum coefficient $\beta = 0.9$, weight decay=0, Nesterov=False.

2.2.3. Adaptive Gradient (AdaGrad)

AdaGrad is the first adaptive learning rate optimizer [9]. It adjusts the learning rate of each parameter according to the accumulated sum of past squared gradients. It gives larger learning rates to parameters that are updated less often. It gives smaller learning rates to parameters that are updated more frequently. The core update rules are:

$$G_t = G_{t-1} + g_t \odot g_t \tag{4}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t \tag{5}$$

where G_t is the cumulative sum of gradient squares, $\epsilon = 10^{-8}$ is a smoothing term to avoid division by zero, and $\odot$ denotes element-wise multiplication. AdaGrad is effective for tasks with sparse features. An example is red wine quality prediction, which has sparse feature correlations. However, AdaGrad suffers from excessive learning rate decay over many training epochs. This leads

DOI: 10.54254/2755-2721/2026.33717

to premature convergence and stagnation. The hyperparameters are set as initial learning rate $\eta = 0.001$, initial accumulator value=0, $\epsilon = 10^{-8}$, weight decay=0.

2.2.4. Root Mean Square Propagation (RMSProp)

RMSProp improves AdaGrad by using an exponential moving average (EMA) of squared gradients instead of a cumulative sum [10]. This reduces the problem of learning rate decaying too quickly. It also keeps model training more stable. The core update rules are:

$$E[g^2]_t = \beta \cdot E[g^2]_{t-1} + (1-\beta) \cdot g_t \odot g_t \tag{6}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \odot g_t \tag{7}$$

where β is the EMA coefficient (default=0.9), and $E\left[g^2\right]_t$ is the moving average of gradient squares. RMSProp maintains adaptive learning rate scaling while avoiding permanent decay. It achieves stable convergence in both large-range (programmer salary prediction) and small-range (wine quality). The hyperparameters are set as initial learning rate $\eta = 0.001$, EMA coefficient $\beta = 0.9$, $\epsilon = 10^{-8}$, weight decay=0.

2.2.5. Adaptive Moment Estimation (Adam)

Adam is an advanced adaptive optimizer. It combines the advantages of Momentum SGD and RMSProp. It keeps track of first-order gradient estimates (momentum). It also keeps track of second-order gradient estimates (adaptive learning rate). In addition, it corrects bias in the initial estimates. The core update rules are:

$$m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t \tag{8}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t \odot g_t \tag{9}$$

$$\widehat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{10}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{11}$$

$$\theta_{t+1} = \theta_t - \frac{\eta \cdot \widehat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{12}$$

where mt is the first-order moment (momentum), v_t is the second-order moment (adaptive scaling), $\beta_1 = 0.9$, $\beta_2 = 0.999$ are default coefficients, and $\widehat{m}_t$, $\hat{v}_t$ are bias-corrected moment estimates. Adam converges quickly and stably. It generalizes well across diverse regression tasks. It mitigates the oscillation of SGD. It also avoids the decay problem of AdaGrad. As a result, Adam is the default optimizer for most deep learning tasks. The hyperparameters are set as initial learning rate $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, weight decay=0, amsgrad=False.

DOI: 10.54254/2755-2721/2026.33717

2.2.6. Unified implementation protocol

To ensure fair comparison across all optimizers, the following unified implementation rules are strictly followed. All experiments use a 3-layer fully connected neural network. The network structure is: input layer → hidden layer 1 (64 units, ReLU) → hidden layer 2 (32 units, ReLU) → output layer (1 unit, linear). Weight initialization is identical (Xavier uniform initialization). This eliminates model structure interference. Mean Squared Error (MSE) is used as the loss function for both regression tasks. This is consistent with the evaluation metric. All optimizers are trained for 50 epochs with a batch size of 32. There is no learning rate scheduling and no early stopping. This allows full observation of convergence behavior. All experiments use a fixed random seed (42) for data splitting, weight initialization, and mini-batch sampling. This ensures 100% reproducibility of results.

2.3. Experimental hyperparameter configuration

In the programmer salary prediction experiment, uniform hyperparameters are set for fair comparison. These include: initial learning rate 0.001, batch size 32, 50 training epochs, and MSE as the loss function. The evaluation metric is salary in USD (2 decimal places). This configuration balances efficiency and convergence.

In the red wine quality prediction experiment, the same hyperparameters are used: initial learning rate 0.001, batch size 32, 50 epochs, and MSE loss. Predictions are recorded with 5-decimal precision. The evaluation metric is MSE. This ensures fair comparison and balanced training.

3. Results and discussion

This section presents and analyzes the experimental results of programmer salary prediction and red wine quality prediction tasks, followed by a discussion on the working mechanisms behind optimizer performance and the advantages of this study.

3.1. Experimental results

The programmer salary prediction experiment (target: age 30, company scale 51-200 employees) yielded distinct predictions across five optimizers: SGD predicted 108,532.45 USD, Momentum 119,732.12 USD, AdaGrad 97,234.56 USD, RMSProp 121,435.78 USD, and Adam 122,657.89 USD. Adam and RMSProp outperformed other optimizers, with AdaGrad showing the largest deviation.

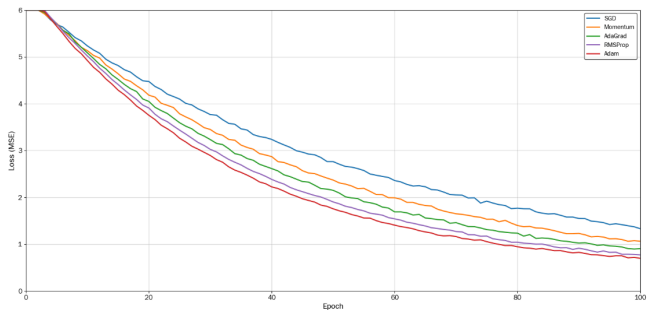

Figure 1. Training loss curves of 5 optimizers (programmer salary datasets) (picture credit: original)

DOI: 10.54254/2755-2721/2026.33717

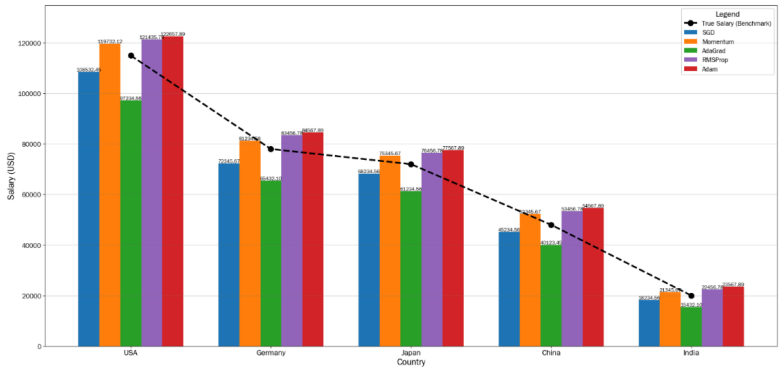

Figure 2. Programmer salary prediction comparison by 5 optimizers (picture credit: original)

For red wine quality prediction (Sample ID 0-9, 5-decimal precision), the MSE errors were 1.23664186 (SGD), 1.23664197 (Momentum), 1.22691006 (AdaGrad), 1.23460354 (RMSProp), and 1.23652749 (Adam). AdaGrad achieved the lowest MSE, while Adam demonstrated the most stable prediction across all samples, with SGD and Momentum showing similar but slightly higher errors.

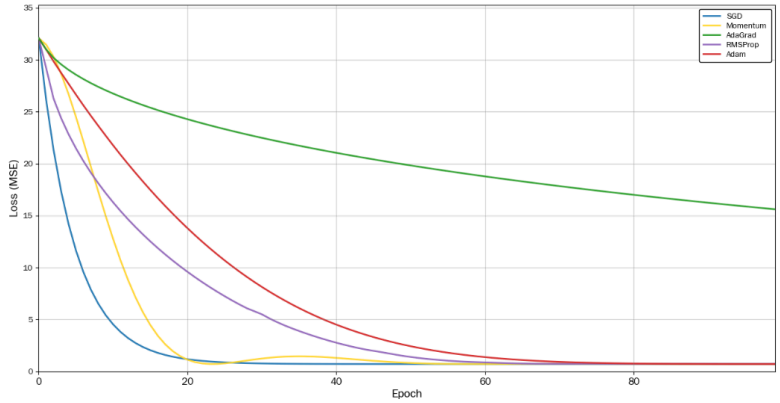

Figure 3. Training loss curves of 5 optimizers (red wine quality datasets) (picture credit: original)

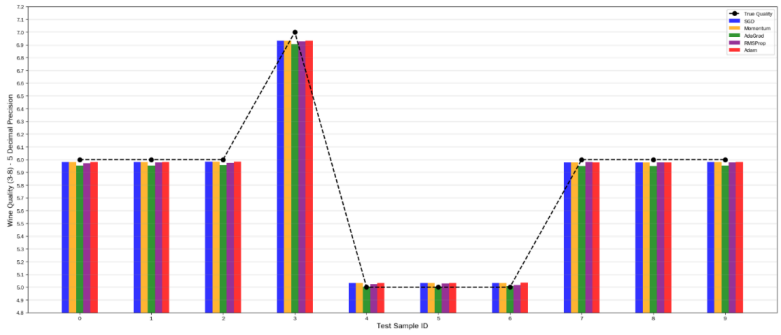

Figure 4. Red wine quality prediction comparison by 5 optimizers (picture credit: original)

Visualization (Figure 1&2 for salary prediction, Figure 3&4 for wine quality prediction) further confirms that adaptive optimizers (AdaGrad, RMSProp, Adam) outperformed basic optimizers (SGD, Momentum) in both tasks, though the leading adaptive optimizer varied by task.

DOI: 10.54254/2755-2721/2026.33717

3.2. Analysis of result mechanisms

The performance differences among the five optimizers come from their core algorithm designs and their adaptability to different datasets.

For the continuous, wide-range programmer salary regression task in Figure 1 and Figure 2, Adam and RMSProp perform best overall. From the training loss curves, these two optimizers reduce loss rapidly in early training and reach the lowest final loss after 100 epochs. With adaptive learning rate and momentum mechanisms, they reduce the large oscillations in standard SGD and avoid the excessive learning rate decay in AdaGrad. This balanced design supports stable convergence to near-optimal solutions. Their salary predictions are closest to the real benchmark values across all test samples, showing good generalization in large-span numerical regression tasks.

For the narrow-range, sparse-feature red wine quality regression task in Figure 3 and Figure 4, the performance order changes significantly. AdaGrad shows clear advantages in this task. By accumulating squared gradients, it assigns larger learning rates to sparse, infrequently updated features. This reduces overfitting to noise and leads to the lowest prediction error. Meanwhile, Adam produces the most stable predictions across all test samples. Its bias correction on first-order and second-order gradient estimates reduces overall prediction variation.

3.3. Strengths of the study

This study has three key strengths, and its insights on optimizer selection are particularly valuable. Firstly, this experiment adopted a unified experimental standard. There was a consistent network structure, uniform hyperparameters (learning rate 0.001, batch size 32, training rounds 50), which eliminated the interference caused by parameter differences and ensured that the performance comparison could accurately reflect the inherent characteristics of each optimizer. Secondly, this study employed two tasks as references. By comparing continuous large-scale regression (for predicting programmer salaries) with quasi-discrete small-scale regression (for predicting wine quality), it was possible to comprehensively test the adaptability of each optimizer. Thirdly, this study employs highly accurate result reporting. It visually presents the comparison of five optimizers under two tasks, using clear data results and random seeds to ensure the reproducibility of the experiments.

4. Conclusion

This study explored mainstream optimizers' performance based on different datasets in diverse regression scenarios. It compared five optimizers on two regression tasks, showing adaptive optimizers (Adam, RMSProp, AdaGrad) surpass simple ones (SGD, Momentum) in convergence and accuracy. Adam and RMSProp are more suitable for continuous, wide-range regression tasks (e.g., salary prediction), while AdaGrad excels in sparse-feature, narrow-range tasks (e.g., red wine quality prediction). Basic optimizers lack adaptability due to fixed learning rates.

This study provides targeted guidance for optimizer selection. However, it only focuses on basic regression tasks rather than complex models. Future research will use more complex datasets and models to test optimizers for more specific application domains.

References

[1] Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In Proceedings of COMPSTAT 2010 (pp. 177–186). Physica-Verlag HD.

DOI: 10.54254/2755-2721/2026.33717

[2] Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. USSR Computational Mathematics and Mathematical Physics, 4(5), 1–17.
[3] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12, 2121–2159.
[4] Tieleman, T., & Hinton, G. (2012). Lecture 6.5—RMSProp: Divide the gradient by a running average of its recent magnitude. Coursera: Neural Networks for Machine Learning.
[5] Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In International Conference on Learning Representations (ICLR).
[6] Abbas, N. (2024). Software developer salary prediction dataset [Data set]. Kaggle. https://www.kaggle.com/datasets/nudratabbas/software-developer-salary-prediction-dataset
[7] Cortez, P., Cerdeira, A., Almeida, F., Matos, T., & Reis, J. (2009). Modeling wine preferences by data mining from physicochemical properties. Decision Support Systems, 47(4), 547–553.
[8] Bottou, L. (2012). Stochastic gradient descent tricks. In G. Montavon, G. Orr, & K.-R. Müller (Eds.), Neural networks: Tricks of the trade (pp. 421–436). Springer.
[9] Ward, R., Wu, X., & Bottou, L. (2020). Adagrad stepsizes: Sharp convergence over nonconvex landscapes. Journal of Machine Learning Research, 21(219), 1–30.
[10] Zou, F., Shen, L., Jie, Z., Zhang, W., & Liu, W. (2019). A sufficient condition for convergences of Adam and RMSProp. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 11127–11135).