Heart Disease Prediction: A Comparative Study of Optimizers' Performance in Deep Neural Networks

Chisom Chibuike[1][0009−0005−5589−814X]⋆ and Adeyinka Ogunsanya[2][0000−0002−4739−5551]

[1] University of Nigeria, Nigeria.
chisom.chibuike.246093@unn.edu.ng
[2] Sydani Institute for Research and Innovation, Sydani Group, Nigeria.
adeyinka.ogunsanya@sydani.org

Abstract. Optimization has been an important factor and topic of interest in training deep learning models, yet less attention has been given to how we select the optimizers we use to train these models. Hence, there is need to dive deeper into how we select the optimizers we use for training and the metrics that determine this selection. In this work, we compare the performance of 10 different optimizers in training a simple Multi-layer Perceptron model using a heart disease dataset from Kaggle. We set up a consistent training paradigm and evaluate the optimizers based on metrics such as convergence speed and stability. We also include some other Machine Learning Evaluation metrics such as AUC, Precision, and Recall, which are central metrics to classification problems. Our results show that there are trade-offs between convergence speed and stability, as optimizers like Adagrad and Adadelta, which are more stable, took longer time to converge. Across all our metrics, we choose RMSProp to be the most effective optimizer for this heart disease prediction task because it offered a balanced performance across key metrics. It achieved a precision of 0.765, recall of 0.827, and an AUC of 0.841, along with faster training time. However, it was not the most stable. We recommend that in less compute constrained environment, this method of choosing optimizers through a thorough evaluation should be adopted to increase the scientific nature and performance in training deep learning models.

Keywords: Deep Neural Network · Optimizer · Stability

1 Introduction

In recent years, Deep Neural Networks (DNNs) have been central to advances in artificial intelligence, driving progress across a wide array of applications, including autonomous systems, medical diagnostics, natural language processing, and

⋆ This work was done while Chisom Chibuike was a research intern at SAIL Innovation Lab, supervised by Adeyinka Ogunsanya.

beyond [10, 20]. Despite their impressive representational power and scalability, the practical effectiveness of DNNs is strongly influenced by the optimization algorithms employed during training [11].

Optimizers serve as the engine of the learning process, updating network parameters to minimize a given loss function. The choice of optimizer directly impacts key aspects of training; convergence speed, stability, generalization, and overall predictive performance [6]. Although a variety of optimizers have been proposed in several literature, including adaptive and nonadaptive methods [2], there is limited systematic literature empirically evaluating their comparative performance in building real-world solutions.

This paper presents a comprehensive empirical study comparing the performance of ten widely used optimization algorithms: Stochastic Gradient Descent (SGD) [9, 16, 18, 21], Stochastic Gradient Descent with Nesterov Momentum (SGD-Nesterov) [24], Root Mean Square Propagation (RMSProp) [25], Adaptive Gradient Algorithm (Adagrad) [5], Adaptive Delta (Adadelta) [29], Adaptive Moment Estimation (Adam) [12], Adam with Decoupled Weight Decay (AdamW) [14], Infinity Norm-based Adam (Adamax) [5], Adaptive Maximum Smoothing Gradient (AMSGrad) [17], and Nesterov-accelerated Adaptive Moment Estimation (Nadam) [4]. Our evaluation is performed on a heart disease dataset from kaggle [22], using a fully connected deep neural network architecture with six hidden layers.

Each optimizer is evaluated across multiple metrics, such as; Convergence Speed, Training Stability (measured via variance in validation loss), and those metrics critical to healthcare prediction tasks; Area Under the ROC Curve (AUC), precision and recall. By focusing on these metrics, we aim to provide a holistic view of the optimizers behavior and nuances, and use this as a guide to selecting the best-performing optimizer, with particular attention to robustness and reliability.

The remainder of this paper is organized as follows: Section 2 reviews related work on optimization in deep learning, with emphasis on comparative studies. Section 3 describes the dataset, preprocessing pipeline, and the architecture of the deep neural network employed. Section 4 presents the experimental results and visualizations comparing the performance of the optimizers. Finally, Section 5 offers concluding remarks and outlines directions for future work, particularly in the context of optimizer selection.

2 Related Works

Several studies have conducted empirical comparisons of optimizers across standard datasets. For instance, [28] evaluated six optimization algorithms; SGD, Nesterov momentum, RMSprop, Adam, Adagrad, and Adadelta on MNIST [3], Fashion-MNIST [26], CIFAR-10 [13], and CIFAR-100 [13]. Their findings suggest that Adam achieved consistently superior test accuracy across all datasets, underscoring the advantage of adaptive methods in image classification tasks.

In a more domain-specific study, Yaqub et al. [27] compared ten optimizers in the context of brain tumor segmentation using the BraTS2015 [15] dataset. Training a convolutional neural network (CNN) architecture on MRI data, they found that Adam outperformed other optimizers, achieving a 99.2% classification accuracy. Their results emphasized the sensitivity of medical imaging models to optimizer selection and hyperparameter configurations.

Other works have explored the interplay of momentum and adaptive learning strategies. For example, [7] studied diabetic detection via backpropagation networks, comparing gradient descent (GD) [19], GD with momentum, GD with adaptive learning rate, and a hybrid approach. They reported that combining momentum and adaptive learning led to faster convergence.

In the specific context of heart disease prediction, García-Ordás et al. [8] applied deep neural networks to the UCI heart disease dataset. They introduced a deep learning-based approach that combines Sparse Autoencoder with a Convolutional Classifier to enhance diagnostic accuracy. Their model, trained using the ADAM optimizer, achieved an accuracy of 90.09%. However, their study primarily focuses on model performance and feature engineering.

Despite several efforts, prior studies often omit evaluating the interplay between training stability and convergence speed, which are vital in evaluating the performance of optimizers in deep learning models, for building real-world applications. Additionally, there remains a lack of focused comparative studies on optimizer behavior in tabular clinical datasets, especially using deep neural networks. Most prior work either targets convolutional architectures or standard benchmarks, leaving a gap in understanding optimizer effectiveness for structured, real-world data.

3 Optimization Algorithms, DNN Architecture, and Datasets

3.1 Optimization Algorithms

In this work, we evaluate the following optimization algorithms: SGD, SGD with Nesterov momentum, RMSprop, Adagrad, Adadelta, Adam, AdamW, Adamax, Nadam, and AMSGrad. *For detailed pseudocode of the optimization algorithms used in this work, please refer to Appendix A.*

3.2 Deep Neural Network Architecture

To ensure a controlled evaluation of the optimization algorithms, we designed a fixed DNN architecture that remained constant throughout the experiments carried out. The model used for the experiment comprises six fully connected hidden layers, structured in an hourglass topology, with neuron sizes $[16, 32, 64, 32, 16, 8]$. Each hidden layer employs the ReLU (Rectified Linear Unit) activation function defined as:

$$\mathrm{ReLU}(z^{(l)}) = \max(0, z^{(l)}) \tag{1}$$

where $z^{(l)} = W^{(l)}h^{(l-1)} + \boldsymbol{b}^{(l)}$ denotes the pre-activation input to the l^{th} hidden layer, $W^{(l)}$ and $\boldsymbol{b}^{(l)}$ are the weight matrix and bias vector of the l^{th} layer, respectively, and $h^{(l-1)}$ is the output from the previous layer.
The output layer contains a single neuron activated by the sigmoid function:

$$\sigma(z^{(L)}) = \frac{1}{1 + e^{-z^{(L)}}} \tag{2}$$

where $z^{(L)} = W^{(L)}h^{(L-1)} + \boldsymbol{b}^{(L)}$ is the pre-activation input to the output neuron. The sigmoid function maps $z^{(L)}$ to the range $[0, 1]$, producing a probability score $\hat{y} \in [0, 1]$ indicating the likelihood of heart disease.

Input and Output Formalism Given an input feature vector $\boldsymbol{x} \in \mathbb{R}^d$, where d is the number of clinical features, the network computes a prediction:

$$\hat{y} = f(\boldsymbol{x}; \theta) \tag{3}$$

where θ represents the set of all trainable parameters (weights and biases). Expanding this, the prediction is computed through a composition of affine transformations and nonlinear activations.

$$\hat{y} = \sigma\left(W^{(L)}\phi^{(L-1)}\left(\cdots\phi^{(1)}\left(W^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right)\cdots\right) + \boldsymbol{b}^{(L)}\right) \tag{4}$$

where $\phi^{(l)}(\cdot)$ denotes the ReLU activation applied at layer l. Here, $L = 7$ denotes the total number of layers (6 hidden + 1 output). [3]

Loss Function The model is trained using the binary cross-entropy loss function:

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\left[y^{(i)}\log\hat{y}^{(i)} + (1 - y^{(i)})\log(1 - \hat{y}^{(i)})\right] \tag{5}$$

where N is the total number of training samples, $y^{(i)} \in \{0, 1\}$ is the ground truth label for the i^{th} sample, and $\hat{y}^{(i)}$ is the predicted probability.

Weight Reinitialization Protocol To ensure that the observed differences in performance are solely attributable to the optimizers under evaluation, we adopted a strict weight reinitialization strategy. The DNN was instantiated once and its initial parameters θ_0 were stored. For every optimizer under evaluation, the model weights were reset to θ_0 prior to training. Formally, for optimizer $\mathcal{O}_j$, training commenced from:

$$\theta_j^{(0)} = \theta_0 \tag{6}$$

This guarantees that each optimizer begins from an identical initialization state, thus isolating the effect of the optimizer parameters on the training dynamics.

[3] The input layer is not counted among the $L = 7$ layers because it does not involve any trainable parameters or transformations. The layer count L includes only those layers that apply affine transformations and nonlinear activations (i.e., hidden and output layers).

Initial Training Setup All models were trained for up to 50 epochs. Metrics such as AUC, precision, recall, convergence speed, and stability (defined as the standard deviation of the validation loss) were recorded per epoch. Hyperparameter tuning was not performed at the initial stage.

Enhanced Training Setup The initial training set-up which was first carried out and the best-performing optimizer was selected for further refinement. In the enhanced training phase, the initial model architecture was trained using the best performing optimizer and augmented with dropout layers to mitigate overfitting. A dropout rate of $p = 0.2$ was applied after each hidden layer. This stage also incorporated early stopping with a patience of 15 epochs, along with a grid search over learning rates $\eta \in \{0.001, 0.01, 0.1\}$ to identify the optimal learning rate. Additionally, a 5-fold cross-validation was employed to ensure robust evaluation and improve the final model's performance.[4]

Our experiment design was kept simple to isolate the effects of optimization algorithms without interference from architectural complexity. Instead of adopting overparameterized or hybrid deep architectures, we implemented a simple six-layer fully connected network to allow a fair comparison of optimizers, ensuring that differences in performance can be attributed primarily to the optimization dynamics rather than structural intricacies.

3.3 The Dataset

Our dataset was sourced from a publicly available Kaggle repository[5] and aggregates patient records relevant to cardiovascular health. It comprises $N = 1190$ samples and $d = 11$ features including the clinical and demographic features summarized in Table 1. The prediction target is a binary label $y \in \{0, 1\}$, where $y = 1$ indicates the presence of heart disease, and $y = 0$, its absence.

Table 1. Overview of the features in the heart disease dataset.

Feature Name	Description
`age`	Age of the patient (in years)
`sex`	Gender of the patient (1 = Male, 0 = Female)
`chest pain type`	Type of chest pain experienced
`resting bps`	Resting blood pressure (mm Hg)
`cholesterol`	Serum cholesterol level (mg/dL)
`fasting blood sugar`	Fasting blood sugar > 120 mg/dL (1 = True, 0 = False)
`resting ecg`	Resting electrocardiographic results
`max heart rate`	Maximum heart rate achieved
`exercise angina`	Exercise-induced angina (1 = Yes, 0 = No)
`oldpeak`	ST depression induced by exercise relative to rest
`ST slope`	Slope of the peak exercise ST segment
`target`	Presence of heart disease (1 = disease, 0 = normal)

[4] The training setup was done to guarantee a principled and reproducible framework for evaluating optimizer performance on a real-world classification task.

[5] https://www.kaggle.com/datasets/sid321axn/heart-statlog-cleveland-hungary-final/data

Exploratory Data Analysis (EDA) EDA revealed that the target variable is relatively balanced, with no significant skew toward either class, eliminating the need for resampling or class weighting. Figure 1 illustrates the distribution of the target labels.

Preprocessing Pipeline To ensure data consistency and prepare for model training, the following preprocessing steps were applied:

1. Missing Values and Redundancies: The dataset contained no missing entries. However, one feature—`fasting blood sugar` was dropped due to low variance ($> 75\%$ of samples held a value of 0). Additionally, 272 duplicate rows were removed.
2. Invalid Zero Entries: Certain physiological measurements (e.g., `cholesterol` and `resting bps`) contained zero values, which are not clinically plausible. These were replaced with the feature-wise mean.
3. Normalization: To reduce sensitivity to outliers, we applied Robust Scaling to all numerical features.
4. Feature Encoding: All categorical features were already numerically encoded. Manual verification confirmed datatype consistency and appropriate class ranges. No additional one-hot encoding was required.
5. Train-Test Split: The dataset was partitioned into training and testing sets using a 70:30 stratified split to preserve class proportions. Within the training set, 20% of data was held out as a validation set during training.

These processes were carried out in an attempt to eradicates any bias from the data in order to ensure that the model is trained on clean, normalized, and representative data while avoiding information leakage or feature imbalance. The resulting dataset provides a reliable foundation for the empirical comparison of optimization algorithms.

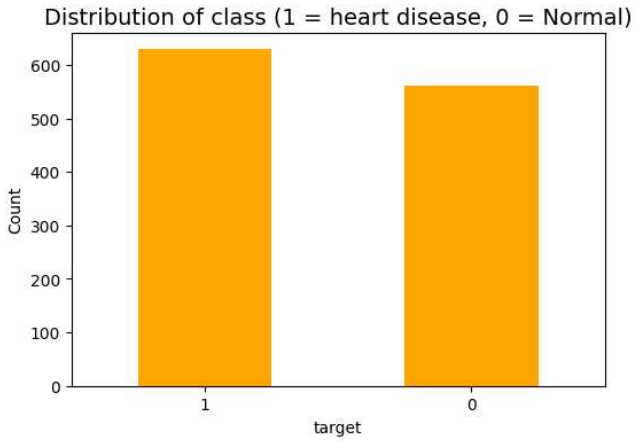

Fig. 1. Distribution of the binary target variable (0 = no disease, 1 = disease).

4 Experimental Results

The evaluation and results focuses on five key metrics: convergence speed, stability, precision, recall, and the Area Under the ROC Curve (AUC-ROC). These metrics are selected to capture both optimizer's dynamics and clinical relevance.

Convergence Speed Convergence speed refers to the number of training epochs required for the model's loss function to stop improving significantly. It indicates how quickly an optimizer leads the model to reach its goal of attaining a minimum of the loss landscape, and is typically measured by the epoch at which validation loss plateaus.

To evaluate convergence speed, we analyzed how quickly each optimizer reduced the training loss to a stable minimum. Specifically, we recorded the number of epochs required for the validation loss to reach its lowest point, referred to as the *convergence epoch*. The results are summarized in Table 2, which lists the final training and validation losses along with the corresponding convergence epoch for each optimizer.
As illustrated in Figure 2, the optimizers exhibited markedly different convergence patterns. **Adam**, **Nadam**, **RMSProp**, **AMSgrad**, and **AdamW** demonstrated rapid convergence within the first 10–18 epochs. However, this speed came at the cost of generalization performance, as we observe notable difference between their training and validation losses, suggesting overfitting.

Table 2. Final training loss, validation loss, and convergence epoch for each optimizer.

Optimizer	Final Training Loss	Final Validation Loss	Convergence Epoch
SGD	0.421812	0.529095	49
Adam	0.114762	1.331397	9
RMSprop	0.157385	0.840259	18
Adagrad	0.662042	0.681975	49
Adadelta	0.697742	0.701515	49
Adamax	0.305551	0.527479	45
Nadam	0.083017	1.239177	15
AMSgrad	0.125040	1.127012	10
AdamW	0.114762	1.331397	9
SGD Nesterov	0.421812	0.529095	49

Stability In this work, we refer to stability, as the standard deviation of the validation loss across training epochs. Optimizers that produce large fluctuations or oscillations in the loss trajectory are deemed less stable, which can adversely affect model reliability and convergence.

To evaluate the stability of each optimizer, we analyzed the variance in validation loss over training epochs. Optimizers that yield minimal fluctuations in

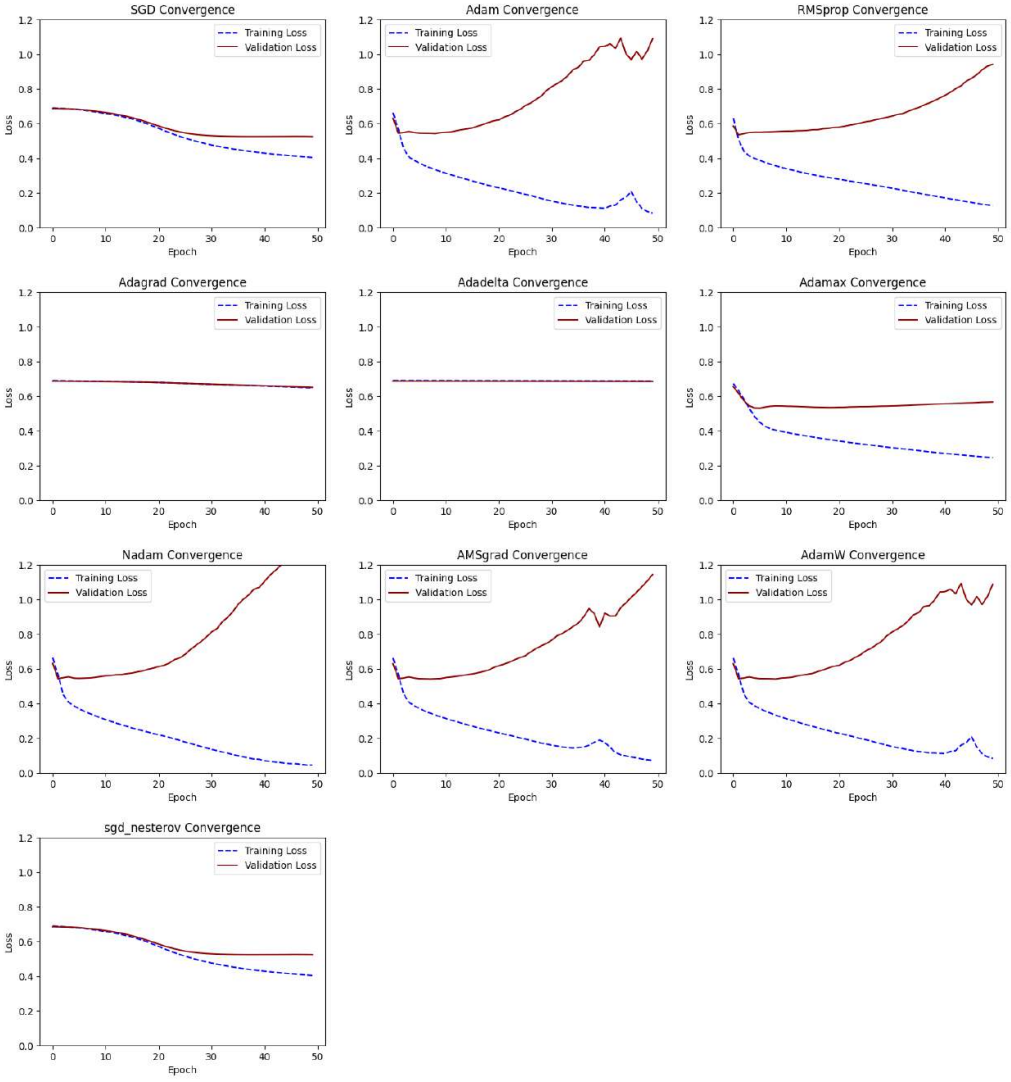

Fig. 2. Convergence curves (training and validation loss) for all optimizers over 50 epochs.

validation loss are considered more stable, because they ensure smoother and more predictable learning. Table 3 presents the standard deviation of the validation loss for each optimizer, serving as a direct measure of training consistency.

Table 3. Stability of each optimizer measured by standard deviation of validation loss.

Optimizer	**Stability (Validation Loss Std. Dev.)**
SGD	0.056739
Adam	0.214864
RMSprop	0.094638
Adagrad	0.005564
Adadelta	0.001403
Adamax	0.047790
Nadam	0.208663
AMSgrad	0.182095
AdamW	0.214864
SGD Nesterov	0.056739

The results in figure 3 and table 3, indicate that; **Adadelta**, **Adagrad**, and **Adamax** demonstrate the highest stability, exhibiting extremely low standard deviations. These optimizers provide consistent training dynamics, making their result more reliable.

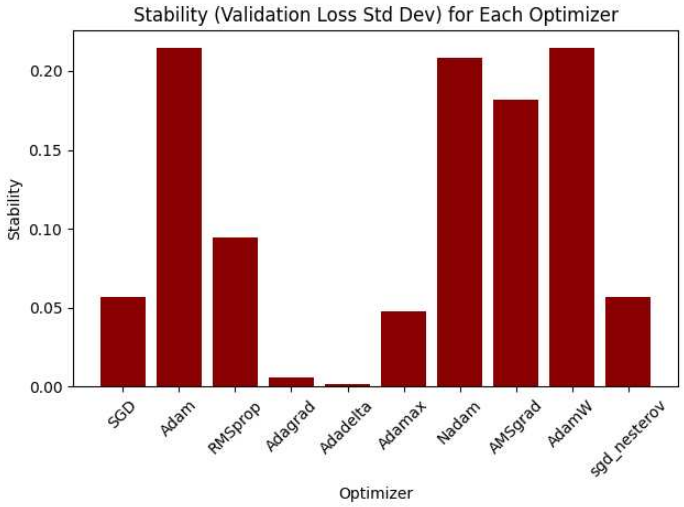

Fig. 3. Standard deviation of validation loss across training epochs for each optimizer.

Precision, Recall, and AUC To assess classification and predictive performance, we evaluated each optimizer on our test set using metrics such as Precision, Recall, and AUC. Table 4 summarizes the final precision, recall, and AUC scores for each optimizer.

Table 4. Final Precision, Recall, and AUC Scores for Each Optimizer on the test set

Optimizer	Precision	Recall	AUC
SGD	0.706	0.800	0.821
Adam	0.707	0.867	0.822
RMSprop	0.765	0.827	0.841
Adagrad	0.449	1.000	0.782
Adadelta	0.449	1.000	0.385
Adamax	0.741	0.800	0.860
Nadam	0.731	0.760	0.843
AMSgrad	0.756	0.787	0.819
AdamW	0.707	0.867	0.822
SGD Nesterov	0.706	0.800	0.821

The results indicate several trade-offs among the optimizers across the evaluated metrics:

1. Precision: **RMSprop** achieved the highest precision (0.765), followed by **AMSgrad** (0.756) and **Adamax** (0.741). High precision indicates that these optimizers were more effective in minimizing false positives, crucial in clinical settings to avoid unnecessary treatment for patients who are not actually at risk.
2. **Recall**: **Adagrad** and **Adadelta** achieved perfect recall (1.000), meaning they correctly identified all true positive cases. However, their low precision (0.449) implies a high rate of false positives. Furthermore, **Adam** and **AdamW** achieved strong recall (0.867), followed by **RMSProp** (0.827) while maintaining higher precision than Adagrad and Adadelta, thus offering a more balanced performance.
3. **AUC**: **Adamax** recorded the highest AUC (0.860), indicating the best overall ability to discriminate between classes across thresholds. **Nadam** (0.843) and **RMSprop** (0.841) followed closely, suggesting their strong generalization and ranking performance.

Based on the comprehensive evaluation across all metrics, **RMSprop** is identified as the most effective optimizer for the heart disease prediction task. RMSprop demonstrates the highest final precision (0.765), a strong AUC score (0.841), and a high recall (0.827), indicating its robust ability to distinguish between positive and negative cases while minimizing false positives and false negatives. Furthermore, RMSprop converges faster, within 18 epochs. Although Adamax achieves the highest AUC (0.860), its recall is comparatively lower (0.800), and its convergence is slower (45 epochs). Overall, RMSprop offers the

best balance across predictive accuracy, generalization, convergence, efficiency, and reliability, making it the most suitable choice for this medical prediction task.

In conclusion, we trained our final model with RMSProp and enhanced it with **dropout regularization**, **hyperparameter tuning**, and **early stopping**, achieving a robust **ROC-AUC score of 92% from the inital 84%**. These results highlight RMSprop as a more suitable optimizer for building a reliable model for this predictive task. Figure 4 illustrates the final Receiver Operating Characteristic (ROC) curve of the model trained with RMSProp, which we identified as the best-performing optimizer. The curve demonstrates a strong area under the curve (AUC = 0.92).

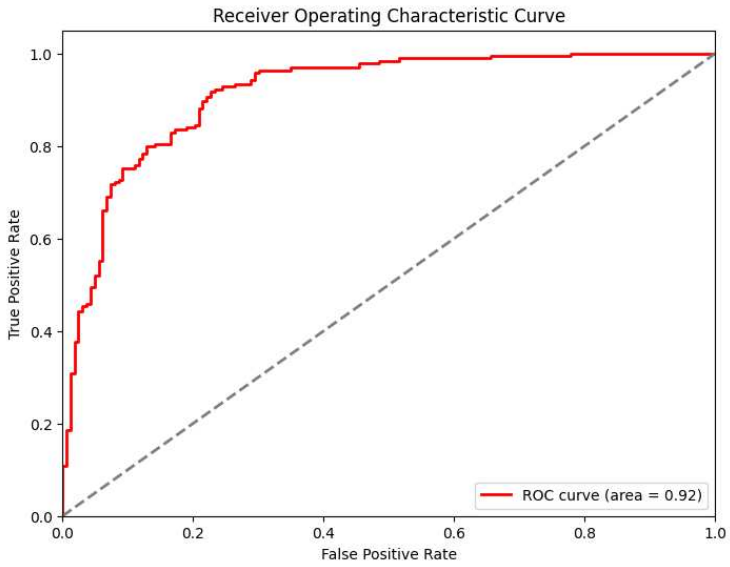

Fig. 4. Final AUC of the overall model trained using RMSProp as the best-performing optimizer.

5 Conclusions, Limitations and Discussion

This study evaluated the impact of various optimization algorithms on the performance of deep neural networks for heart disease prediction. Our findings revealed that; **RMSprop** is the most effective optimizer for this heart disease prediction task, offering a balanced performance across key metrics. It achieved strong precision (0.765), high recall (0.827), and a solid AUC (0.841), along with faster training time. However, it is not very stable. In addition, we noticed the trade-off between convergence speed and stability, as optimizers like Adagrad

and Adadelta which are more stable, took longer time to converge. By incorporating dropout regularization, hyperparameter tuning, the best optimizer for this task, and early stopping, we further enhanced the model's ability to generalize, achieving a final ROC-AUC score of 92%.

These results emphasize the role of optimizers in shaping the outcome of deep learning models. Consequently, we recommend that researchers adopt a deliberate and systematic approach to evaluating optimization algorithms when designing predictive systems. Optimizers are not merely plug-and-play components as they substantially influence model training dynamics, convergence speed, and overall predictive performance. A careful selection informed by empirical evidence can lead to more accurate and robust models.

Nonetheless, our study is not without limitations. The dataset used (both in terms of its size and structure) and the model's architecture, may have influenced the performance outcomes of the optimizers evaluated. Future work should investigate the generalizability of these findings by conducting experiments on larger datasets and different deep learning architectures. Such efforts will help in establishing more reliable benchmarks and uncovering optimizer behaviors that may only emerge at scale. Further more, future work should focus on investigating the reason behind the trade-offs between the convergence speed and stability of the optimizers used in this study.

Finally, we emphasize the need for the research community to continue developing optimization techniques that are more stable, adaptive, and capable of handling the increasing complexity and diversity of real-world data. Since no single optimizer currently performs optimally across all tasks and datasets, advancing the state of optimization remains a fundamental challenge in machine learning research.

5.1 Ethical Consideration

The dataset used in this study is a publicly available health dataset obtained from the Kaggle repository[6] and also published by [23]. It was compiled by merging multiple pre-existing datasets, including the Cleveland [1] heart disease datasets. According to the dataset description, all personally identifiable information has been removed, and the data has been anonymized. To the best of our knowledge, the dataset does not contain any sensitive personal health information, and its use complies with ethical standards for secondary health data.

References

1. Bhatnagar, R.: Heart disease - cleveland. https://www.kaggle.com/datasets/ritwikb3/heart-disease-cleveland (2020)

[6] https://www.kaggle.com/datasets/sid321axn/heart-statlog-cleveland-hungary-final/data

2. de Souza, C.P.G., Kurka, P.R.G., Lins, R.G., de Araújo, J.M.: Performance comparison of non-adaptive and adaptive optimization algorithms for artificial neural network training applied to damage diagnosis in civil structures. Applied Soft Computing **104**, 107254 (2021). https://doi.org/10.1016/j.asoc.2021.107254
3. Deng, L.: The mnist database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine **29**(6), 141–142 (2012)
4. Dozat, T.: Incorporating Nesterov Momentum into Adam. In: Proceedings of the 4th International Conference on Learning Representations. pp. 1–4 (2016)
5. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research **12**(61), 2121–2159 (2011), http://jmlr.org/papers/v12/duchi11a.html
6. Elharrouss, O., Mahmood, Y., Bechqito, Y., Serhani, M.A., Badidi, E., Riffi, J., Tairi, H.: Loss functions in deep learning: A comprehensive review (2025), https://arxiv.org/abs/2504.04242
7. Endah, S.N., Widodo, A.P., Fariq, M.L., Nadianada, S.I., Maulana, F.: Beyond back-propagation learning for diabetic detection: Convergence comparison of gradient descent, momentum and adaptive learning rate. 2017 1st International Conference on Informatics and Computational Sciences (ICICoS) pp. 189–194 (2017), https://api.semanticscholar.org/CorpusID:6586789
8. García-Ordás, M.T., Bayón-Gutiérrez, M., Benavides, C., Aveleira-Mata, J., Benítez-Andrades, J.A.: Heart disease risk prediction using deep learning techniques with feature augmentation. Multimedia Tools and Applications pp. 1–15 (2023), https://api.semanticscholar.org/CorpusID:257661996
9. Hardt, M., Recht, B., Singer, Y.: Train faster, generalize better: Stability of stochastic gradient descent. In: International conference on machine learning. pp. 1225–1234. PMLR (2016)
10. Herath, H.: Evolution and advancements from neural network to deep learning (03 2025). https://doi.org/10.13140/RG.2.2.13671.36009
11. Huang, L., Qin, J., Zhou, Y., Zhu, F., Liu, L., Shao, L.: Normalization techniques in training dnns: Methodology, analysis and application (2020), https://arxiv.org/abs/2009.12836
12. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization (2017), https://arxiv.org/abs/1412.6980
13. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Tech. Rep. 0, University of Toronto, Toronto, Ontario (2009), https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf
14. Loshchilov, I., Hutter, F.: Fixing weight decay regularization in adam. CoRR **abs/1711.05101** (2017), http://arxiv.org/abs/1711.05101
15. Menze, B.H., Jakab, A., Bauer, S., Kalpathy-Cramer, J., Farahani, K., Kirby, J., Burren, Y., Porz, N., Slotboom, J., Wiest, R., Lanczi, L., Gerstner, E., Weber, M.A., Arbel, T., Avants, B.B., Ayache, N., Buendia, P., Collins, D.L., Cordier, N., Corso, J.J., Criminisi, A., Das, T., Delingette, H., Demiralp, C., Durst, C.R., Dojat, M., Doyle, S., Festa, J., Forbes, F., Geremia, E., Glocker, B., Golland, P., Guo, X., Hamamci, A., Iftekharuddin, K.M., Jena, R., John, N.M., Konukoglu, E., Lashkari, D., Mariz, J.A., Meier, R., Pereira, S., Precup, D., Price, S.J., Raviv, T.R., Reza, S.M.S., Ryan, M., Sarikaya, D., Schwartz, L., Shin, H.C., Shotton, J., Silva, C.A., Sousa, N., Subbanna, N.K., Szekely, G., Taylor, T.J., Thomas, O.M., Tustison, N.J., Unal, G., Vasseur, F., Wintermark, M., Ye, D.H., Zhao, L., Zhao, B., Zikic, D., Prastawa, M., Reyes, M., Van Leemput, K.: The multimodal brain tumor image segmentation benchmark (brats). IEEE Transactions on Medical Imaging **34**(10), 1993–2024 (2015). https://doi.org/10.1109/TMI.2014.2377694

16. Nemirovski, A., Yudin, D.: On cezari's convergence of the steepest descent method for approximating saddle point of convex-concave functions. In: Soviet Mathematics. Doklady. vol. 19, pp. 258–269 (1978)
17. Reddi, S.J., Kale, S., Kumar, S.: On the convergence of adam and beyond. CoRR **abs/1904.09237** (2019), http://arxiv.org/abs/1904.09237
18. Robbins, H., Monro, S.: A stochastic approximation method. The annals of mathematical statistics pp. 400–407 (1951)
19. Ruder, S.: An overview of gradient descent optimization algorithms (2017), https://arxiv.org/abs/1609.04747
20. Schmidhuber, J.: Deep learning in neural networks: An overview. Neural Networks **61** (04 2014). https://doi.org/10.1016/j.neunet.2014.09.003
21. Shalev-Shwartz, S., Singer, Y., Srebro, N.: Pegasos: Primal estimated sub-gradient solver for svm. In: Proceedings of the 24th international conference on Machine learning. pp. 807–814 (2007)
22. Sid321axn: Heart statlog: Cleveland and hungary combined. https://www.kaggle.com/datasets/sid321axn/heart-statlog-cleveland-hungary-final/data (2022), accessed: 2025-05-20
23. Siddhartha, M.: Heart disease dataset (comprehensive) (2020). https://doi.org/10.21227/dz4t-cm36
24. Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in deep learning. In: Dasgupta, S., McAllester, D. (eds.) Proceedings of the 30th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 28, pp. 1139–1147. PMLR, Atlanta, Georgia, USA (17–19 Jun 2013), http://proceedings.mlr.press/v28/sutskever13.html
25. Tieleman, T., Hinton, G.: Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning (2012)
26. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms (2017), https://arxiv.org/abs/1708.07747
27. Yaqub, M., Feng, J., Zia, M.S., Arshid, K., Jia, K., Rehman, Z.U., Mehmood, A.: State-of-the-art cnn optimizer for brain tumor segmentation in magnetic resonance images. Brain Sciences **10**(7) (2020). https://doi.org/10.3390/brainsci10070427
28. Zaheer, R., Shaziya, H.: A study of the optimization algorithms in deep learning. In: 2019 Third International Conference on Inventive Systems and Control (ICISC). pp. 536–539 (2019). https://doi.org/10.1109/ICISC44355.2019.9036442
29. Zeiler, M.D.: Adadelta: An adaptive learning rate method (2012), https://arxiv.org/abs/1212.5701

A Appendix

Algorithm 1 Stochastic Gradient Descent (SGD)

Require: Training data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$, learning rate η, batch size b, maximum iterations T
Ensure: Optimized parameters θ_T
Initialize parameters θ_0 randomly
for $t = 1$ **to** T **do**
  Sample a mini-batch $B_t = \{(x^{(i)}, y^{(i)})\}_{i=1}^{b}$ from $\mathcal{D}$
  Compute gradient: $g_t = \nabla_\theta J(\theta_t; B_t)$
  Update parameters: $\theta_{t+1} = \theta_t - \eta \cdot g_t$
end for
return θ_T

Algorithm 2 SGD with Nesterov Momentum

Initialize parameters θ_0, learning rate η, momentum coefficient $\mu \in [0, 1)$, and velocity vector $v_0 \leftarrow 0$
Set iteration $t \leftarrow 0$
while not converged **do**
  Compute look-ahead position: $\tilde{\theta}_t = \theta_t - \mu v_t$
  Evaluate gradient at look-ahead: $g_t = \nabla_\theta J(\tilde{\theta}_t; B_t)$
  Update momentum: $v_{t+1} = \mu v_t + \eta g_t$
  Update parameters: $\theta_{t+1} = \theta_t - v_{t+1}$
  Increment $t \leftarrow t + 1$
end while
return final parameters θ_t

Algorithm 3 Adam

Initialize parameters θ_0, first moment $m_0 \leftarrow 0$, second moment $v_0 \leftarrow 0$, timestep $t \leftarrow 0$
while not converged **do**
  Increment timestep: $t \leftarrow t + 1$
  Compute gradient: $g_t = \nabla_\theta J(\theta_{t-1}; B_t)$
  Update biased first moment estimate: $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  Update biased second moment estimate: $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
  Compute bias-corrected moments:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

  Update parameters:

$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

end while
return final parameters θ_t

Algorithm 4 Adamax Optimization Algorithm

Initialize parameters θ_0, first moment $m_0 \leftarrow 0$, infinity norm $u_0 \leftarrow 0$, timestep $t \leftarrow 0$
while not converged **do**
  Increment timestep: $t \leftarrow t + 1$
  Compute gradient: $g_t = \nabla_\theta J(\theta_{t-1}; B_t)$
  Update first moment estimate: $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  Update infinity norm estimate: $u_t = \max(\beta_2 u_{t-1}, |g_t|)$
  Compute update: $\Delta\theta_t = -\frac{\eta}{u_t + \epsilon} m_t$
  Update parameters: $\theta_t = \theta_{t-1} + \Delta\theta_t$
end while
return final parameters θ_t

Algorithm 5 AdamW Optimization Algorithm

Initialize parameters θ_0, first moment $m_0 \leftarrow 0$, second moment $v_0 \leftarrow 0$, timestep $t \leftarrow 0$
while not converged **do**
  Increment timestep: $t \leftarrow t + 1$
  Compute gradient: $g_t = \nabla_\theta J(\theta_{t-1}; B_t)$
  Update biased first moment: $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  Update biased second moment: $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
  Compute bias-corrected moments:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

  Update parameters with weight decay:

$$\theta_t = \theta_{t-1} - \alpha \cdot \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda \theta_{t-1} \right)$$

end while
return final parameters θ_t

Algorithm 6 Adagrad Optimization Algorithm

Initialize parameters θ_0, accumulator $r_0 \leftarrow 0$, timestep $t \leftarrow 0$
while not converged **do**
  Increment timestep: $t \leftarrow t + 1$
  Compute gradient: $g_t = \nabla_\theta J(\theta_{t-1}; B_t)$
  Update accumulator: $r_t = r_{t-1} + g_t^2$
  Update parameters:

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{r_t} + \epsilon} \cdot g_t$$

end while
return final parameters θ_t

Algorithm 7 RMSprop Optimization Algorithm

Initialize parameters θ_0, squared gradient accumulator $r_0 \leftarrow 0$, timestep $t \leftarrow 0$
while not converged **do**
  Increment timestep: $t \leftarrow t + 1$
  Compute gradient: $g_t = \nabla_\theta J(\theta_{t-1}; B_t)$
  Update exponential moving average of squared gradients:

$$r_t = \beta r_{t-1} + (1 - \beta) g_t^2$$

  Update parameters:

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{r_t} + \epsilon} \cdot g_t$$

end while
return final parameters θ_t

Algorithm 8 Nadam Optimization Algorithm

Initialize parameters θ_0, first moment $m_0 \leftarrow 0$, second moment $v_0 \leftarrow 0$, timestep $t \leftarrow 0$
Set hyperparameters: learning rate η, decay rates $\beta_1, \beta_2 \in [0, 1)$, small constant ϵ
while not converged **do**
  Increment timestep: $t \leftarrow t + 1$
  Compute gradient: $g_t = \nabla_\theta J(\theta_{t-1}; B_t)$
  Update biased first moment estimate:

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

  Update biased second moment estimate:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

  Compute Nesterov-accelerated update:

$$\tilde{g}_t = \frac{\eta}{\sqrt{v_t} + \epsilon} \left(\beta_1 m_t + \frac{(1 - \beta_1) g_t}{1 - \beta_1^t} \right)$$

  Update parameters:

$$\theta_t = \theta_{t-1} - \tilde{g}_t$$

end while
return final parameters θ_t

Algorithm 9 AMSGrad Optimization Algorithm

Initialize parameters θ_0, first moment $m_0 \leftarrow 0$, second moment $v_0 \leftarrow 0$, maximum second moment $\hat{v}_0 \leftarrow 0$, timestep $t \leftarrow 0$
while not converged **do**
  Increment timestep: $t \leftarrow t + 1$
  Compute gradient: $g_t = \nabla_\theta J(\theta_{t-1}; B_t)$
  Update first moment estimate: $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$
  Update second moment estimate: $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
  Update maximum second moment: $\hat{v}_t = \max(\hat{v}_{t-1}, v_t)$
  Update parameters:

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \cdot m_t$$

end while
return final parameters θ_t

Algorithm 10 Adadelta Optimization Algorithm

Initialize parameters θ_0, accumulated squared gradient $E[g^2]_0 \leftarrow 0$, accumulated squared updates $E[\Delta\theta^2]_0 \leftarrow 0$, timestep $t \leftarrow 0$
while not converged **do**
  Increment timestep: $t \leftarrow t + 1$
  Compute gradient: $g_t = \nabla_\theta J(\theta_{t-1}; B_t)$
  Update accumulated squared gradients:

$$E[g^2]_t = \rho E[g^2]_{t-1} + (1 - \rho) g_t^2$$

  Compute parameter update:

$$\Delta\theta_t = -\frac{\sqrt{E[\Delta\theta^2]_{t-1} + \epsilon}}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_t$$

  Update accumulated squared updates:

$$E[\Delta\theta^2]_t = \rho E[\Delta\theta^2]_{t-1} + (1 - \rho) \Delta\theta_t^2$$

  Update parameters:

$$\theta_t = \theta_{t-1} + \Delta\theta_t$$

end while
return final parameters θ_t