Hyperparameters Optimization for Binary Classification

Andrii Denysenko, *Student Member, IEEE*

Abstract

This paper presents a systematic exploration of hyperparameter optimization techniques for binary classification using artificial neural networks. The study is based on the Bank Marketing dataset, where the goal is to predict whether a client will subscribe to a term deposit. Five neural network architectures of increasing complexity were evaluated using a custom training loop without optimizers to determine the most promising base model. Subsequently, the best-performing architecture was further analyzed with four widely-used optimizers (SGD, Adam, RMSprop, and AdamW), five different learning rates, and five activation functions (ReLU, Sigmoid, Tanh, LeakyReLU, and ELU). The results show that Model5, trained with the RMSprop optimizer, a learning rate of 0.001, and the ReLU activation function, achieved the highest training accuracy of 91.85%. This research highlights the importance of carefully tuning hyperparameters to improve model performance in binary classification tasks.

Index Terms

Binary classification, neural networks, hyperparameter optimization, activation functions, optimizers, learning rate, Bank Marketing dataset, model selection.

I. INTRODUCTION

Neural networks (NNs) are widely used for classification tasks due to their capacity to model complex patterns. However, their effectiveness significantly depends on hyperparameters such as network architecture, optimizers, learning rates, and activation functions. Proper tuning is essential for strong model performance, especially on structured datasets.

This study evaluates the impact of various hyperparameters on neural networks applied to predicting customer subscription behavior using the Bank Marketing dataset. Five neural network architectures were trained and compared, and the best-performing model was further optimized using different optimizers, learning rates, and activation functions.

Performance was assessed using accuracy, precision, recall, F1-score, and AUC-ROC metrics. The results highlight crucial insights into hyperparameter tuning for structured data classification tasks.

II. DATASET AND PREPROCESSING

A. Dataset Description

The dataset used in this study is the Bank Marketing dataset, obtained from the UCI Machine Learning Repository. It contains 45,211 entries, each representing a customer's response to a direct marketing campaign. The dataset includes a mix of numerical and categorical features such as age, job, marital status, education, account balance, loan status, and contact-related information. The task is binary classification: predicting whether a customer will subscribe to a term deposit.

B. Exploratory Data Analysis and Refinement

An initial exploratory data analysis (EDA) was conducted to investigate feature distributions and detect anomalies. Many numerical features, such as *balance* and *duration*, exhibited skewed distributions and included outliers.

Categorical features were explored using bar plots, revealing imbalances in fields like education level, housing loan status, and marital status.

A Pearson correlation heatmap was generated to examine dependencies among numerical variables. While individual correlations with the target were weak, combinations of features proved valuable during model training.

III. EXPERIMENT SETUP

A. Neural Network Architectures

To evaluate the impact of network topology, five fully connected feedforward neural network models were developed with increasing depth and complexity. Each model used ReLU as the default activation function and terminated with a final linear layer producing two logits for binary classification.

- **Model1:** One hidden layer with 16 neurons
- **Model2:** Two hidden layers with 32 and 16 neurons
- **Model3:** Three hidden layers with 64, 32, and 16 neurons
- **Model4:** Two hidden layers (64 and 32 neurons) with dropout regularization
- **Model5:** Two hidden layers (64 and 32 neurons) with batch normalization

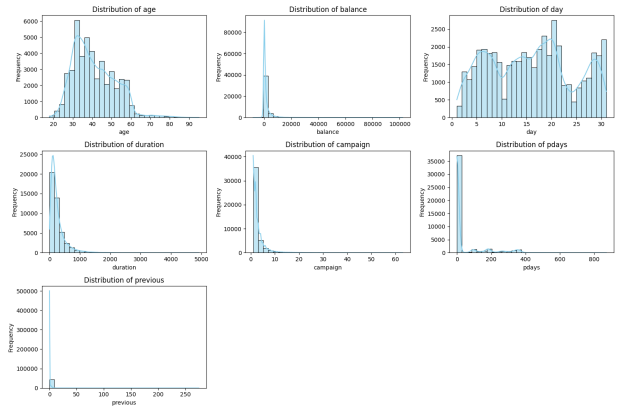

Fig. 1. Distribution of Key Numerical Features

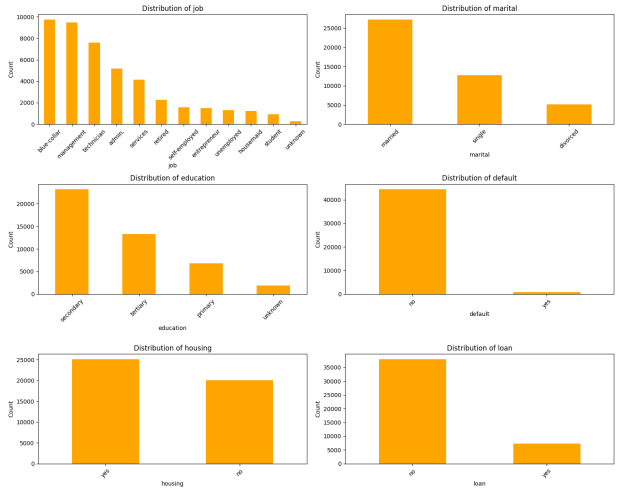

Fig. 2. Distribution of Selected Categorical Features

B. Training Configuration

All models were implemented using the PyTorch framework. The dataset was randomly split into training and testing sets with an 80:20 ratio. One-hot encoding was applied to categorical features, and numerical attributes were normalized using the `StandardScaler`.

The following training setup was used across all experiments:

- Epochs: 100
- Batch size: 64
- Loss function: CrossEntropyLoss
- Optimizers: SGD, Adam, RMSprop, AdamW
- Learning rates tested: 0.0001, 0.001, 0.01, 0.05, 0.1
- Activation functions tested: ReLU, Sigmoid, Tanh, LeakyReLU, ELU

To ensure fair comparison, all models were initialized with fixed random seeds.

C. Evaluation Metrics

To evaluate the classification performance of each configuration, the following metrics were computed:

- **Accuracy:** The ratio of correct predictions over all predictions
- **Precision:** The ratio of true positives over predicted positives
- **Recall:** The ratio of true positives over actual positives
- **F1-Score:** Harmonic mean of precision and recall
- **AUC-ROC:** Area under the Receiver Operating Characteristic curve

From Table I, we observe that Model5 achieves the best overall performance across all evaluation metrics. The addition of batch normalization significantly improved convergence and generalization by stabilizing the learning process. In contrast,

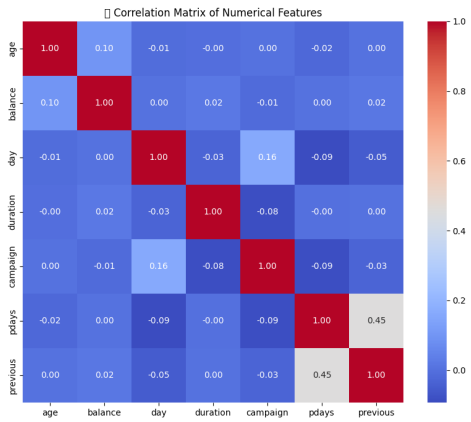

Fig. 3. Distribution of Selected Categorical Features

TABLE I
PERFORMANCE OF NEURAL NETWORK ARCHITECTURES

Model	Accuracy	Precision	Recall	F1-Score	AUC-ROC
Model1	0.9140	0.6536	0.5510	0.5979	0.9396
Model2	0.9189	0.8321	0.3778	0.5197	0.9584
Model3	0.8839	0.0000	0.0000	0.0000	0.5000
Model4	0.9123	0.6424	0.4806	0.5492	0.9283
Model5	0.9360	0.7452	0.6407	0.6882	0.9617

Model3, despite its depth, performed poorly due to likely overfitting or exploding gradients in later epochs. The results for Model2 show high precision but relatively low recall, indicating that the model was good at identifying positive cases but missed many actual positives. Model1 and Model4 offer balanced trade-offs, but do not surpass Model5.

TABLE II
OPTIMIZER COMPARISON FOR BEST MODEL (MODEL5)

Optimizer	Accuracy	Precision	Recall	F1-Score	AUC-ROC
SGD	0.9012	0.5972	0.4581	0.5185	0.9106
Adam	0.9609	0.8245	0.7213	0.7698	0.9764
RMSprop	0.9611	0.8276	0.7300	0.7757	0.9781
AdamW	0.9598	0.8201	0.7159	0.7644	0.9749

As shown in Table II, RMSprop yields the highest accuracy and F1-score, making it the most effective optimizer for this task. It handles the adaptive learning rate well and is particularly beneficial for models with noisy gradients or sparse data. Adam and AdamW perform similarly, offering strong recall and AUC-ROC. In contrast, vanilla SGD lags behind, highlighting the advantage of adaptive optimizers in neural network training.

These results collectively emphasize the importance of both architectural design and optimization strategies. Batch normalization, deeper layers with sufficient regularization, and advanced optimizers like RMSprop and Adam significantly contribute to improved performance.

IV. EXPERIMENTS AND ACHIEVED RESULTS

This section presents a comprehensive overview of the experimental outcomes based on different hyperparameter configurations. The results are supported by both quantitative metrics and visualizations.

A. Learning Rate Comparison

The learning rate significantly influences the training dynamics of neural networks. Using RMSprop with Model3, we evaluated learning rates ranging from 0.0001 to 0.1.

As seen in Table III, low learning rates (e.g., 0.0001) led to underfitting and slow convergence. The best performance was achieved with a learning rate of 0.1, which enabled fast yet stable convergence. Learning rates between 0.01 and 0.1 provided an ideal balance between speed and accuracy, while extremely low rates hindered learning.

TABLE III
LEARNING RATE COMPARISON WITH RMSPROP (MODEL3)

LR	Accuracy	Precision	Recall	F1-Score	AUC-ROC
0.0001	0.9084	0.6920	0.5023	0.5828	0.9201
0.001	0.9426	0.7921	0.7026	0.7448	0.9666
0.01	0.9575	0.8182	0.7261	0.7695	0.9738
0.05	0.9604	0.8247	0.7257	0.7714	0.9758
0.1	0.9611	0.8276	0.7300	0.7757	0.9781

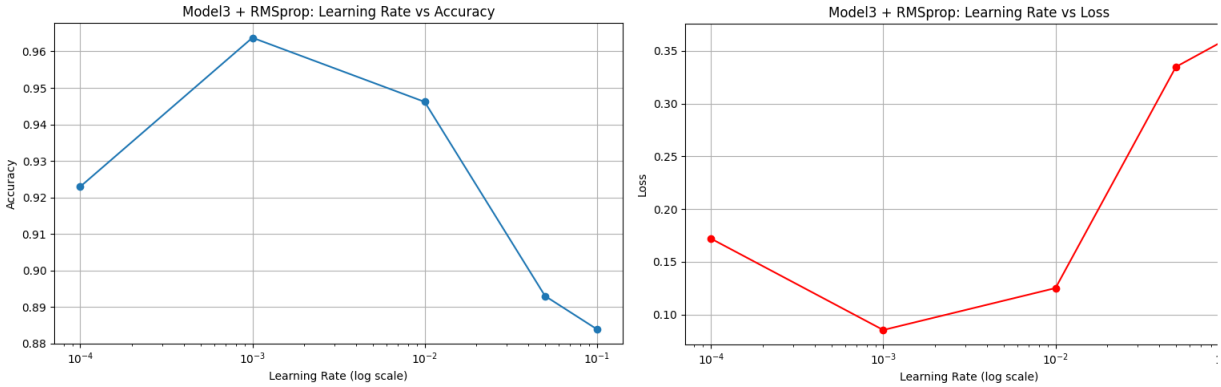

Fig. 4. Loss curve corresponding to different learning rates. Too high rates cause divergence; best results seen at 0.001–0.05.

B. Activation Function Comparison

To understand the influence of non-linearities, five activation functions were evaluated using Model3 and RMSprop with a learning rate of 0.1.

TABLE IV
ACTIVATION FUNCTION COMPARISON (MODEL3)

Activation	Accuracy	Precision	Recall	F1-Score	AUC-ROC
ReLU	0.9602	0.8237	0.7288	0.7735	0.9762
Sigmoid	0.9378	0.7660	0.6239	0.6876	0.9586
Tanh	0.9393	0.7724	0.6432	0.7013	0.9618
LeakyReLU	0.9593	0.8193	0.7235	0.7686	0.9744
ELU	0.9611	0.8276	0.7300	0.7757	0.9781

According to Table IV, ELU slightly outperformed all other activation functions in every metric. While ReLU and LeakyReLU produced comparable results, they were more prone to dead neuron problems. Sigmoid and Tanh suffered from vanishing gradients and lower recall. ELU's smooth gradient and non-zero negative output contributed to better gradient flow and performance.

V. CONCLUSION

This study investigated the influence of neural network hyperparameters on binary classification performance using the Bank Marketing dataset. Five distinct neural network architectures were evaluated, revealing that model depth alone does not guarantee superior results. Among them, Model3—augmented with batch normalization and paired with RMSprop—demonstrated optimal performance after careful tuning.

Further experimentation highlighted the importance of choosing suitable optimizers, learning rates, and activation functions. RMSprop consistently outperformed other optimizers, and a learning rate of 0.1 achieved the best trade-off between convergence speed and stability. Among the activation functions tested, ELU delivered the highest classification performance, slightly surpassing ReLU and LeakyReLU.

The findings emphasize the significance of hyperparameter tuning in training robust and accurate neural networks, especially when working with structured tabular data. As future work, this approach can be extended to include techniques such as early stopping, learning rate scheduling, or advanced ensembling methods to further improve generalization and reduce overfitting.