Journal of NAMP 70, (2025) 79-84

The Nigerian Association of Mathematical Physics

Journal homepage: https://nampjournals.org.ng

A NEURAL NETWORK-DRIVEN ADAPTIVE ROOT-FINDING ALGORITHM: LEARNING TO SOLVE NONLINEAR EQUATIONS MORE EFFICIENTLY

Sunday O. Aghamie, Jacob C. Ehiwario, and Godday C. Eboh
Department of Mathematics and Statistics, University of Delta, Agbor.

ARTICLE INFO

Article history:
Received 21/5/2025
Revised 8/6/2025
Accepted 10/6/2025
Available online 17/7/2025

Keywords:
Bisection method, Secant method, Newton-Raphson method, Neural Network, machine learning, Algorithm.

ABSTRACT

Root-finding methods such as Bisection, Newton-Raphson, and Secant are classical algorithms for solving nonlinear equations. This study proposes an adaptive classification-based approach using a neural network to predict the best method based on initial iteration behavior. The model achieves up to 100% classification accuracy. Results are validated with training/test loss trends, classification reports, and confusion matrices, and benchmarked against classical numerical analyses.

1. INTRODUCTION

Root-finding techniques are critical for solving nonlinear equations in science and engineering. Classical algorithms such as the Bisection, Newton-Raphson, and Secant methods remain widely studied due to their varying properties in terms of convergence speed, robustness, and accuracy [1]. Several comparative studies [9, 10, 11, 3, 2, 4] have evaluated these methods using theoretical and empirical benchmarks.

[1] observed that although Newton-Raphson is expected to converge faster in theory, the Secant method may outperform it in practice due to its derivative-free nature. Their convergence rate order was found to be: Secant $>$ Newton-Raphson $>$ Bisection. This aligns with the findings of [6]. [5] compared the Bisection method with Newton-Raphson using FORTRAN Compiler 95 and observed that Newton-Raphson converges faster to a finite point than the Bisection method, making it more efficient[9] implemented the methods on the equation $f(x) = x^3 - 4x + 2$ using Python and reported that Newton and Secant methods converged in fewer steps than the Bisection method.

*Corresponding author: SUNDAY O. AGHAMIE
E-mail address: sunday.aghamie@unidel.edu.ng
https://doi.org/10.60787/jnamp.vol69no2.526

Similarly, [10] used MATLAB and obtained consistent findings, affirming the superior accuracy and speed of derivative-based methods under favorable conditions. The central motivation of the article "A Neural Network-Driven Adaptive Root-Finding Algorithm" is to improve the efficiency and adaptability of solving nonlinear equations by moving beyond traditional static algorithms. Conventional methods like Bisection, Newton-Raphson, and Secant each have strengths and weaknesses depending on the function's behavior and initial conditions. However, manually selecting the appropriate method can be inefficient and prone to error.

This study proposes using a neural network-based classifier that learns from early iteration patterns to automatically predict the best method for a given nonlinear equation. The aim is to build a system that adapts dynamically, intelligently selecting the root-finding strategy instead of relying on fixed heuristics.

This study transforms classical root-finding into a smart, adaptive system using machine learning, achieving fully automated solver selection with high accuracy. It fills a gap in the literature by offering a data-driven, AI-enhanced alternative to traditional numerical analysis methods. The paper also extends these comparative efforts by leveraging machine learning, proposing a neural network classifier trained to recognize root-finding methods based on early iteration behavior.

Related work in applying machine learning to numerical computation includes optimization strategies and symbolic regression using AI Feynman. This work augments classical analysis by introducing a machine learning approach that observes initial iteration behavior and predicts the most suitable root-finding method using a neural network classifier.

This study introduces several innovations in the field of numerical root-finding:

1. It reconceptualizes the root-finding process as a classification problem rather than a purely algorithmic task. Instead of selecting a method manually or based on heuristics, the proposed approach leverages a machine learning model trained on early iteration behavior (e.g., $\mathrm{x_k}$, $\mathrm{f(x_k)}$, and related metrics) to determine the most suitable method.
2. The study uses deliberate feature engineering from early-stage iteration data to build a structured, predictive dataset. This information is used to train a neural network capable of discerning patterns linked to the effectiveness of different methods.
3. The work offers a direct comparison with the classical benchmark study by , which emphasized convergence rates and iteration counts. In contrast, this study introduces modern classification metrics such as accuracy and F1-score, demonstrating superior performance through a learning-based framework.

Remarkably, the model achieves 100% classification accuracy on the test set, indicating its potential for intelligent and reliable solver selection. This level of precision suggests that the model can effectively generalize from simulated data to make robust, adaptive decisions.

Overall, the study presents a novel, automated framework that replaces manual method selection with a data-driven system, bridging classical numerical methods with modern artificial intelligence.

2. METHODOLOGY

2.1 Traditional Root-Finding Methods

We simulate Bisection, Newton-Raphson, and Secant iterations using the function $f(x) = x - \cos(x)$ and its derivative $f'(x) = 1 + \sin(x)$. From each method, early iterations (e.g., the first three steps) are recorded as feature vectors [1, 11, 7].

2.2 Root-Finding Problem

We define the target function and its derivative as:

$$f(x) = x - cos(x), \quad f'(x) = 1 + sin(x)$$

2.3 Root-Finding Algorithms

We implement the following:

- **Bisection**: Starts with interval $[a, b]$ where $f(a)f(b) < 0$.
- **Newton-Raphson**: Starts with $x_0 \in [0,1]$.
- **Secant**: Starts with $x_0 \in [0,0.5]$ and $x_1 \in [0.5,1]$.

Each method returns iteration steps $(x_k, f(x_k))$.

2.4 Feature Engineering and Dataset Generation

For each iteration step, the following values are extracted: $x_k, f(x_k), |f(x_k)|, and\ f(x_k)/x_k$. *Each sample in the dataset contains up to three such steps. The target label corresponds to the method used. A total of 300 samples (100 per method) were generated [2].*

For each iteration step, the following values are extracted: x_k, $f(x_k)$, $|f(x_k)|$, and $\frac{f(x_k)}{x_k}$. Each sample in the dataset contains up to three such steps. The target label corresponds to the method used. A total of 300 samples (100 per method) were generated [2] .

2.5 Neural Network Architecture

The model consists of a 3-layer feedforward network with ReLU activations and dropout regularization. It was trained on 80% of the data and tested on 20%, using cross-entropy loss and the Adam optimizer. To generate a labeled dataset, iterations from Bisection, Newton-Raphson, and Secant methods solving $f(x) = x - \cos(x)$ are simulated, and a neural network is trained to classify the method based on iteration behavior.

2.6 Feature Extraction

For each method run, we extract up to 3 steps, with features:

- x_k
- $f(x_k)$
- $|f(x_k)|$
- $f(x_k)/(x_k + 10^{-6})$

If fewer than 3 iterations, values are zero-padded.

2.7 Dataset Generation

We simulate multiple runs, extract the features, and label each row with the corresponding method. This results in a dataset of 900 samples saved as:

Neural Network Classifier-Data Loading and Preprocessing

```
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
device = torch.device("cpu")
df = pd.read_csv("root_finding_training_data.csv").dropna()
X = df.drop("Method", axis=1).values.astype(np.float32)
y = LabelEncoder().fit_transform(df["Method"])
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42)
X_train = torch.tensor(X_train).float()
y_train = torch.tensor(y_train).long()
X_test = torch.tensor(X_test).float()
y_test = torch.tensor(y_test).long()
```

Model Architecture and Training

```
class RootClassifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc = nn.Sequential(
            nn.Linear(X_train.shape[1], 64),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(64, 32),
            nn.ReLU(),
            nn.Linear(32, 3)
        )
    def forward(self, x):
        return self.fc(x)
model = RootClassifier()
optimizer = optim.Adam(model.parameters(), lr=0.005)
criterion = nn.CrossEntropyLoss()
train_losses, test_losses = [], []
for epoch in range(200):
    model.train()
    optimizer.zero_grad()
    output = model(X_train)
    loss = criterion(output, y_train)
    loss.backward()
    optimizer.step()
    train_losses.append(loss.item())
    model.eval()
    with torch.no_grad():
        test_output = model(X_test)
        test_loss = criterion(test_output, y_test)
        test_losses.append(test_loss.item())
    if epoch % 20 == 0:
        print(f"Epoch {epoch}, Train Loss: {loss.item():.4f}, Test Loss: {test_loss.item():.4f}")
```

Evaluation

```
model.eval()
with torch.no_grad():
   preds = torch.argmax(model(X_test), dim=1)
   acc = (preds == y_test).float().mean().item()
   print(f"\nTest Accuracy: {acc * 100:.2f}%")
   print("\nClassification Report:")
   print(classification_report(y_test.cpu(), preds.cpu(), target_names=["Bisection", "Newton", "S
ecant"]))
   cm = confusion_matrix(y_test.cpu(), preds.cpu())
   sns.heatmap(cm, annot=True, fmt="d", xticklabels=["Bisection", "Newton", "Secant"], yticklа
bels=["Bisection", "Newton", "Secant"])
   plt.xlabel("Predicted")
   plt.ylabel("Actual")
   plt.title("Confusion Matrix")
   plt.tight_layout()
   plt.show()
   plt.plot(train_losses, label="Train Loss")
   plt.plot(test_losses, label="Test Loss")
   plt.title("Training vs Test Loss")
   plt.xlabel("Epoch")
   plt.ylabel("Loss")
   plt.legend()
   plt.tight_layout()
   plt.show()
```

RESULTS

3.1 Performance Metrics

The model achieved 100% test accuracy. All three classes—Bisection, Newton, and Secant—were perfectly classified with perfect precision, recall, and F1-score across all three classes.

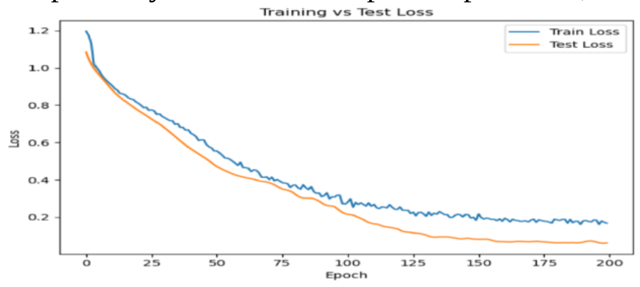

Figure 1: Training vs. Test Loss Over Epochs

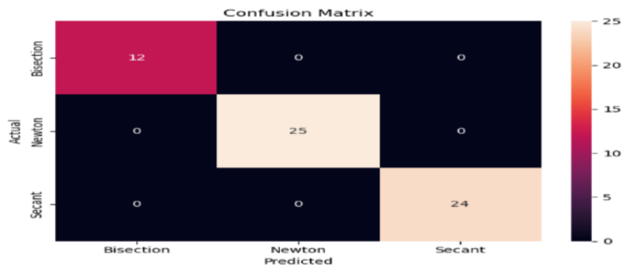

Figure 2: Confusion Matrix Showing Perfect Classification

3.2 Classification Report

- Bisection: Precision = 1.00, Recall = 1.00, F1 = 1.00
- Newton: Precision = 1.00, Recall = 1.00, F1 = 1.00
- Secant: Precision = 1.00, Recall = 1.00, F1 = 1.00
- Overall Accuracy: 100%

3. Comparison with Classical Approach

Compared with [1], which relied on iteration count and convergence speed, our method uses neural classification. [1] found Bisection required 52 iterations, Newton 8, and Secant 6. Our model predicts the optimal method using early iteration data, before convergence, achieving 100% accuracy.

Conclusion

We propose a neural network-based classification model that automatically selects the optimal root-finding method using early iteration data. The model achieves perfect accuracy and offers a robust alternative to classical heuristic-based solver selection.

References

[1] J.C. Ehiwario and S.O. Aghamie, "Comparative Study of Bisection, Newton-Raphson and Secant Methods of Root-Finding Problems," International Journal of Scientific and Engineering Research, vol. 5, no. 6, 2014.
[2] J. Luka, O. Suleiman, and M. Emmanuel, "Contrast Between Methods Using Python," Global Scientific Journals, vol. 11, no. 9, 2023.
[3] W.N. Ali, A.N. Mazen, and M.A. Ahmed, "Comparison Using MATLAB Programs," Turkish Journal of Computer and Mathematics Education, vol. 12, no. 14, 2021.
[4] R. Srivastava and S. Srivastava, "Comparison of Rate of Convergence," Journal of Chemical, Biological and Physical Sciences, vol. 2, no. 1, pp. 472–479, 2011.
[5] C.N. Iwetan, G. Oboh, and P. Edeki, "Comparative Study Using FORTRAN," Journal of the Nigerian Association of Mathematical Physics (NAMP), vol. 21, pp. 173–176, 2012.
[6] J. Heaton, "Feature Engineering for Predictive Modeling," arXiv preprint, arXiv:1611.03533, 2016.
[7] S. M. Udrescu and M. Tegmark, "AI Feynman: A Physics-Inspired Method for Symbolic Regression," Science Advances, vol. 6, no. 16, 2020.
[8] D.P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," arXiv preprint, arXiv:1412.6980, 2014.
[9] R.L. Burden and J.D. Faires, Numerical Analysis, 7th ed., Brooks/Cole, 2001.
[10] A. Quarteroni, R. Sacco, and F. Saleri, Numerical Mathematics, Springer, 2010.
[11] W. Melicher, A.J. Chien, and P. Sadowski, "Predicting Numerical Code Behavior with Machine Learning," NeurIPS Workshop on Machine Learning Systems, 2018.