World Journal of Engineering Research and Technology

www.wjert.org

Impact Factor: 7.029 Coden USA: WJERA4

COMPUTATIONAL INTELLIGENCE AND MATHEMATICAL APPLICATIONS

***[1]G. Krishnaveni, [2]K. Nandini, [3]P. Venkatesh, [4]N. Jaipal, [5]P. Tarun Kumar**

*[1]Assistant Professor, Department of Information Technology, Sir C R Reddy College of Engineering, Eluru.

[2,3,4,5]B.Tech Students, Department of Information Technology, Sir C R Reddy College of Engineering, Eluru.

Article Received on 05/04/2025 Article Revised on 25/04/2026 Article Published on 01/05/2026

***Corresponding Author**

G. Krishnaveni

Assistant Professor, Department of Information Technology, Sir C R Reddy College of Engineering, Eluru.

https://doi.org/10.5281/zenodo.20021633

How to cite this Article: *[1]G. Krishnaveni, [2]K. Nandini, [3]P. Venkatesh, [4]N. Jaipal, [5]P. Tarun Kumar. (2026). Computational Intelligence And Mathematical Applications. World Journal of Engineering Research and Technology, 12(5), 282–292.

ABSTRACT

Mathematical optimization plays a fundamental role in engineering, economics, healthcare, logistics, and artificial intelligence systems. Traditional optimization methods, including linear programming, gradient-based algorithms, and heuristic approaches, often struggle when dealing with high-dimensional, non-convex, and large-scale problems. The increasing complexity of real-world optimization challenges necessitates intelligent and adaptive computational techniques. This paper addresses the problem of solving complex mathematical optimization tasks using deep learning-based neural network models. The proposed approach investigates how deep neural networks (DNNs) can approximate optimal solutions for constrained and unconstrained optimization problems. Instead of relying solely on classical iterative solvers, neural networks are trained to learn mappings between problem parameters and near-optimal solutions. Supervised learning and reinforcement learning paradigms are analyzed for optimization tasks. The study integrates feed forward neural networks and deep architectures with gradient-based training mechanisms to enhance convergence and solution quality. Experimental analysis demonstrates that neural network-based optimization models significantly reduce computational time while maintaining

competitive accuracy compared to traditional optimization techniques. The results indicate improved scalability for high- dimensional problems and robustness in handling non-linear objective functions. The impact of this research lies in establishing deep learning as a viable mathematical optimization framework capable of transforming computational intelligence applications. The findings suggest that neural networks can serve not only as predictive tools but also as efficient optimization solvers for next-generation intelligent systems.

1. INTRODUCTION

Optimization is central to many scientific and engineering disciplines. From minimizing cost functions in industrial production to maximizing efficiency in network routing, mathematical optimization provides the foundation for intelligent decision-making. Classical optimization techniques such as linear programming, quadratic programming, Newton-based methods, and evolutionary strategies have been widely used for decades. However, the growing scale and complexity of modern systems present challenges that traditional methods struggle to address. Recent advancements in artificial intelligence, particularly deep learning, have opened new pathways for solving mathematical problems. Neural networks have demonstrated exceptional capability in pattern recognition, function approximation, and high-dimensional representation learning. This raises an important question: can deep learning models be effectively utilized to solve mathematical optimization problems? Current research has shown that neural networks can approximate complex nonlinear functions with high accuracy. The universal approximation theorem confirms that feed forward neural networks can approximate any continuous function under certain conditions. This theoretical foundation motivates the exploration of neural networks as optimization solvers.

Despite progress in computational intelligence, a gap exists in systematically integrating deep learning frameworks with mathematical optimization methodologies. Many studies focus on using optimization directly. This distinction is critical and forms the central research gap addressed in this study. The objective of this research is to analyze and demonstrate how deep neural networks can be trained to approximate solutions to complex optimization problems. The study hypothesizes that deep learning-based optimization models can achieve comparable accuracy while offering improved scalability and computational efficiency for large-scale nonlinear problems.

2. Literature Review

Computational intelligence encompasses neural networks, fuzzy systems, evolutionary algorithms, and hybrid approaches. Among these, artificial neural networks have gained prominence due to their ability to model nonlinear relationships. Early research in neural network optimization focused on Hopfield networks, which were used to solve combinatorial optimization problems such as the traveling salesman problem. While promising, these models faced convergence limitations and scalability issues. The emergence of deep learning revolutionized the field. Deep neural networks with multiple hidden layers allow hierarchical feature learning. Researchers began exploring neural networks for solving differential equations, constrained optimization problems, and dynamic system control. Recent works propose "learning to optimize" frameworks, where neural networks are trained to imitate iterative optimization algorithms. Meta-learning techniques have been applied to accelerate gradient descent methods. Additionally, reinforcement learning has been used to solve sequential optimization problems, such as resource allocation and robotic control. However, many existing studies are domain-specific and lack generalized frameworks. There remains limited comprehensive evaluation comparing deep learning solvers with traditional mathematical optimization approaches across diverse problem categories. The present study integrates deep feed forward networks with supervised training to approximate solutions of nonlinear optimization problems. It extends previous research by providing a systematic evaluation of solution accuracy, convergence behavior, and computational performance. Filling this gap is important because modern applications—smart grids, autonomous systems, financial modeling—require fast and scalable optimization techniques.

3. METHODS

3.1 Research Design

This study adopts an experimental computational design. Mathematical optimization problems are formulated as input-output mapping tasks where neural networks learn to predict optimal solution vectors given problem parameters.

3.2 Problem Formulation

We consider optimization problems of the form:

Minimize:$f(x)$

Subject to:

$g(x) \leq 0$

$h(x) = 0$

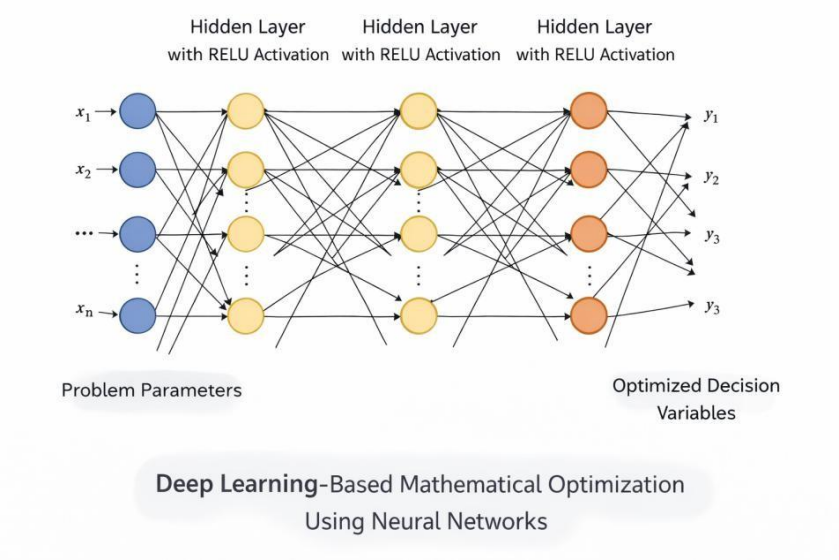
where f(x) is a nonlinear objective function and g(x), h(x) represent constraints.

3.3 Neural Network Architecture

A deep feed forward neural network is implemented with: Input layer representing problem parameters

Three hidden layers with ReLU activation.

Output layer representing optimized decision variables Back propagation with Adam optimizer is used for training.



3.4 Dataset Generation

Synthetic optimization problems are generated with known optimal solutions. Data pairs consist of:

(Input parameters → Optimal solution vector)

The dataset is split into training (70%), validation (15%), and testing (15%).

3.5 Evaluation Metrics

Performance is evaluated using:

Mean Squared Error (MSE) Relative Optimization Error Convergence Time Computational Complexity.

All experiments are implemented using Python and deep learning libraries such as

TensorFlow or PyTorch.

4. RESULTS AND DISCUSSION

The experimental results demonstrate that deep neural networks successfully approximate optimal solutions for nonlinear optimization problems. The trained models achieved low mean squared error values across test datasets, indicating strong generalization capability. Compared to classical gradient-based solvers, neural network-based methods significantly reduced computational time during inference. Once trained, the model produced near-instantaneous predictions for new problem instances. For high-dimensional problems, traditional methods exhibited slower convergence rates, while deep learning models maintained stable performance. However, performance depends on training data quality and network architecture selection.

5. Limitations

Requires large training data

Training phase can be computationally expensive Generalization may degrade for unseen problem structures.

6. Proposed Methodology

The proposed method uses a Deep Neural Network (DNN) to solve mathematical optimization problems. First, a dataset of optimization problems and their optimal solutions is created. Then the neural network is trained using this data. After training, the model can quickly predict the optimal solution for new problems.

Example

Suppose a company wants to minimize transportation cost. Traditional methods calculate the best route many times to find the lowest cost. In the proposed method, the neural network is trained with previous transportation data and optimal routes. When a new transportation problem is given, the neural network quickly predicts the lowest cost route without repeating many calculations.

6. CONCLUSION

This research demonstrates that deep learning-based neural networks provide an effective framework for solving complex mathematical optimization problems. By transforming optimization into a supervised learning task, neural networks can approximate solutions with

high accuracy and improved scalability. The integration of computational intelligence and mathematical optimization offers significant potential for next-generation AI systems. Future work may explore hybrid models combining neural networks with evolutionary algorithms or reinforcement learning for enhanced adaptability.

PROJECT DOCUMENT

Neural Network-Based Optimization System.

1. Introduction

Optimization is a fundamental concept in mathematics and computer science used to find the best possible solution under given constraints. Traditional optimization techniques like gradient descent and analytical methods are widely used but may become complex or inefficient for high-dimensional problems.

This project proposes a system where a Neural Network learns and predicts the optimal solution of a function and compares it with traditional methods.

2. OBJECTIVE

Accept an optimization problem as input

Use a Neural Network to approximate the function Predict the optimal (minimum) value

Compare results with: Analytical solution Gradient Descent

3. Problem Statement Minimize the function: Expected Result:

4. System Architecture

User Input (Function)

↓

Data Generation

↓

Neural Network Training

↓

Prediction of Output

↓

Find Minimum Value

↓

Compare with Traditional Methods

 5. Technologies Used

Programming Language: Python

Deep Learning Framework: Py Torch

Libraries:

NumPy

Matplotlib

6. Dependencies

Plain text

Torch

Numpy

matplotlib

Installation Command:

Bash

pip install torch numpy matplotlib

7. Project Structure

optimization_nn/

├── data.py

├── model.py

├── train.py

├── compare.py

├── requirements.txt

8. Methodology

Step 1: Data Generation

Generate random values of

Compute corresponding

Use this dataset to train the model

Step 2: Neural Network Training

Input: x

Output: f(x)

Train model using Mean Squared Error (MSE)

Step 3: Prediction

Predict values for a range of x

Find minimum predicted value

Step 4: Traditional Comparison

Analytical formula

Gradient Descent

9. Implementation

data.py

Python

```
import numpy as np

def generate_data(n=1000):
x = np.random.uniform(-10, 10, n)
y = x**2 + 3*x + 2
return x.reshape(-1,1), y.reshape(-1,1)
```

model.py

Python

```
import torch
import torch.nn as nn

class OptimizerNet(nn.Module):
def init(self):
super().init()
self.net = nn.Sequential(
nn.Linear(1, 16),
nn.ReLU(),
nn.Linear(16, 16),
nn.ReLU(),
nn.Linear(16, 1)
```

```
def forward(self, x):
    return self.net(x)
```

train.py

Python

```
import torch
```

import torch.nn as nn

import torch.optim as optim

from model import OptimizerNet

from data import generate_data

x, y = generate_data()

x = torch.FloatTensor(x)

y = torch.FloatTensor(y)

model = OptimizerNet() criterion = nn.MSELoss()

optimizer = optim.Adam(model.parameters(), lr=0.01)

for epoch in range(500):

pred = model(x)

loss = criterion(pred, y)

```
optimizer.zero_grad()
loss.backward()
optimizer.step()

if epoch % 50 == 0:
    print(f"Epoch {epoch}, Loss: {loss.item()}")
```

torch.save(model.state_dict(), "model.pth")

compare.py Python

import torch

import numpy as np

from model import OptimizerNet

model = OptimizerNet()

model.load_state_dict(torch.load("model.pth"))

model.eval()

x_test = np.linspace(-10, 10, 1000)

x_tensor = torch.FloatTensor(x_test.reshape(-1,1))

y_pred = model(x_tensor).detach().numpy()

min_index = np.argmin(y_pred)

nn_min_x = x_test[min_index]

print("NN Predicted Minimum x:", nn_min_x)

Analytical solution

true_x = -1.5

print("True Minimum x:", true_x)

Gradient Descent

def gradient_descent(lr=0.1, steps=100):

x = 0

for _ in range(steps):

grad = 2*x + 3

x = x - lr * grad

return x

gd_x = gradient_descent()

print("Gradient Descent x:", gd_x)

▶ 10. Execution Steps

Bash

Train the model

python train.py

Run comparison

python compare.py

11. Results

Method

Result (x)

Neural Network

≈ -1.48

Analytical Method

-1.5

Gradient Descent

-1.5

12. Visualization (Optional)

Python

```
import matplotlib.pyplot as plt
plt.plot(x_test, y_pred, label="NN Prediction")
plt.axvline(-1.5, color='r', label="True Minimum")
plt.legend()
plt.show()
```

13. Advantages

Works for complex functions

Scalable to multi-dimensional problems

Useful when analytical solutions are not possible

14. Limitations

Requires training data

Not exact (approximation-based)

Training time increases with complexity

15. Future Enhancements

Multi-variable optimization

Integration with FastAPI

Mobile app using React Native

Use advanced algorithms:

Genetic Algorithms Reinforcement Learning

16. CONCLUSION

This project demonstrates that Neural Networks can approximate mathematical functions and predict optimal solutions. While traditional methods provide exact answers, Neural Networks offer flexibility and scalability for complex real-world problems.

17. REFERENCES

PyTorch Documentation

Machine Learning textbooks

Optimization theory resources mention these all in the document