Decoding Digits: Building a Neural Network for Handwritten Digit Classification

A NumPy-Only Approach to Handwritten Digit Classification Using Neural Networks

1st Vaibhav Shewale
B. Tech Artificial intelligence and data science
Ajeenkya DY Patil University
Pune, India
vaibhav.shewale@adypu.edu.in

2nd Vaishnavi Suryawanshi
B. Tech Artificial intelligence and data science
Ajeenkya DY Patil University
Pune, India
vaishnavi.suryawanshi@adypu.edu.in

***Abstract*—This report outlines the development of a neural network designed to classify handwritten digits using only NumPy for all numerical computations. The model is trained and validated on the MNIST dataset sourced from Kaggle. Featuring a simple feedforward architecture, the implementation achieves an accuracy of approximately 83 on validation data. The primary aim is to provide a clear understanding of neural network fundamentals by building each component manually, including data processing, activation functions, and optimization using gradient descent.**

***Index Terms*—MNIST, Neural Network, NumPy, Image Classification, Gradient Descent, Supervised Learning, Python Implementation**

I. Introduction

Digit classification is an essential task in the field of machine learning and image processing. The MNIST dataset, consisting of labeled images of handwritten digits, provides a standardized challenge for benchmarking classification models. This project focuses on constructing a complete neural network from scratch using NumPy, enabling in-depth learning of the fundamental components such as forward propagation, loss calculation, and backpropagation without relying on high-level frameworks.

II. Background and Related Work

A. The Idea of using only Numpy.

Various advanced frameworks such as TensorFlow and PyTorch offer highly optimized solutions for neural network development. However, such libraries often abstract the underlying mechanics of learning. The purpose of this project is educational, aiming to reinforce core concepts by implementing every layer, activation function, and training step manually. Comparable work includes introductory machine learning courses and textbook examples that break down these core operations.

Identify applicable funding agency here. If none, delete this.

III. Methodology

The methodology for building the digit recognition model using NumPy is broken down into eight key steps, from data preparation to training the network. Each step is carefully designed to reinforce the theoretical and practical aspects of machine learning.

A. Dataset Preparation

- **Source**: Kaggle MNIST Digit Recognizer competition
- **Description**: Each image is 28x28 pixels, flattened into 784 features
- **Normalization**: Pixel values are scaled from 0–255 to 0–1 for consistent input
- **Label Encoding**: Output labels are converted to one-hot encoded vectors

B. Model Structure

- **Input Layer**: 784 neurons (one for each pixel)
- **Hidden Layer**: 64 neurons, chosen for a balance of simplicity and performance
- **Output Layer**: 10 neurons, one for each digit class

C. Activation Functions

The activation function of a node in an artificial neural network is a function that calculates the output of the node based on its individual inputs and their weights. Nontrivial problems can be solved using only a few nodes if the activation function is nonlinear.

- **Sigmoid Function**:

```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

- **Softmax Function**:

```
def softmax(z):
    exp_z = np.exp(z - np.max(z))
    return exp_z / np.sum(exp_z, axis=0)
```

D. Forward Propagation

- Compute layer inputs and activations

```
Z = np.dot(W, A_prev) + b
A = sigmoid(Z)  # or softmax(Z) for final layer
```

E. Loss Function

- Cross-Entropy Loss used for classification tasks

```
def cross_entropy_loss(Y, Y_hat):
    m = Y.shape[1]
    return -np.sum(Y * np.log(Y_hat)) / m
```

F. Backpropagation

- Compute gradients to update weights and biases

```
W_grad = np.dot(dZ, A_prev.T) / m
b_grad = np.sum(dZ, axis=1, keepdims=True) / m
```

G. Training Loop

- Combine forward pass, loss computation, backpropagation, and weight update

```
for epoch in range(num_epochs):
    A, Y_hat = forward_propagation(X)
    loss = cross_entropy_loss(Y, Y_hat)
    dA = backward_propagation(Y, Y_hat)
    update_parameters(W, b, dA)
```

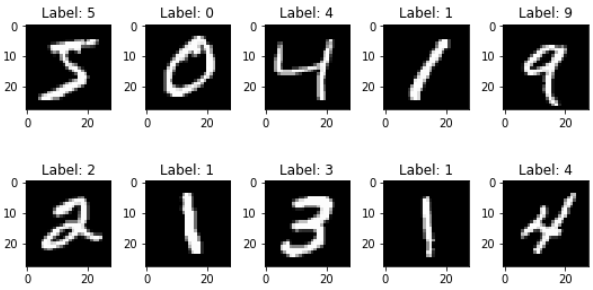
H. Evaluation and Predictions

- After training the model is tested with unseen data.

```
def predict(X):
    A = forward_propagation(X)
    return np.argmax(A, axis=0)
```

I. Experimental Setup

- **Learning Rate**: 0.01
- **Training Epochs**: 100
- **Optimizer**: Standard gradient descent
- **Initialization**: Random normal distribution
- **Train-Test Split**: 80/20 ratio



1. Training Metrics:

- Final Training Accuracy: 83%
- Validation Accuracy: 82~
- Loss Trend: Decreasing loss over epochs, indicating effective learning.

IV. Discussion

The implemented model shows a promising performance given its simplicity and lack of optimization techniques such as momentum or adaptive learning rates. The accuracy achieved is a strong indicator that even basic architectures, when implemented correctly, can yield respectable results on structured datasets. This foundational approach also offers flexibility for extending architecture into deeper or more complex forms in the future.

V. Conclusion

This project effectively demonstrates the core principles of neural networks using a ground-up approach. The final model achieves satisfactory results while providing detailed insight into each operational step of training and prediction. This work serves as a starting point for further exploration into deep learning methodologies and network architectures.

ACKNOWLEDGEMENT

The dataset used in this project was provided by Kaggle. Tools such as Jupyter Notebook and NumPy were instrumental in developing and testing the implementation.

References

[1] Y. LeCun et al., "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] Kaggle, "Digit Recognizer," [Online]. Available: https://www.kaggle.com/competitions/digit-recognizer

[3] IEEE Editorial Style Manual, [Online]. Available: https://journals.ieeeauthorcenter.ieee.org/your-role-in-article-production/ieee-editorial-style-manual/

[4] S. Raschka, *Python Machine Learning*, Packt Publishing Ltd., 2015.

[5] M. Nielsen, "Neural Networks and Deep Learning," [Online]. Available: http://neuralnetworksanddeeplearning.com/

[6] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.