Arithmetic Operator Recognition Using a Single-Layer Artificial Neural Network

Submitted by:

PIYUSH SINGH
Roll No: 524EC0014

Submitted to:

Prof. Mohammad Asan Basiri
Department of Electronics and Communication Engineering

Indian Institute of Information Technology Design and Manufacturing, Kurnool

26 January 2026

1 Problem Statement

The objective of this assignment is to design a Single Layer Artificial Neural Network (ANN) to detect and classify four arithmetic symbols from images: Addition (+), Subtraction (−), Multiplication (×), and Division (÷). The network must be trained using the Least Mean Square (LMS) algorithm, and the weights must be updated iteratively to minimize classification error.

2 Methodology

2.1 Network Architecture

We implemented a single-layer perceptron with the following specifications:

- **Input Layer:** 32×32 pixel grayscale images. These are flattened into a vector of size 1024. A bias unit (+1) is appended, resulting in 1025 input nodes.
- **Output Layer:** 4 neurons, where each neuron represents one class (+, −, ×, ÷).
- **Weights:** A weight matrix of size 1025×4.

2.2 Mathematical Model (LMS Algorithm)

The network uses the Sigmoid activation function:

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

To ensure stability and high accuracy, we utilized the Gradient Descent rule including the derivative of the Sigmoid function (often referred to as the Delta Rule for non-linear units). The weight update formula used is:

$$W_{new} = W_{old} + \eta \cdot (Target - Output) \cdot [Output(1 - Output)] \cdot Input^T \tag{2}$$

Where:

- η (Eta) is the learning rate (set to 0.05).
- $Output(1 - Output)$ is the derivative of the Sigmoid function.

3 Dataset Generation

A script was written using Python and OpenCV to generate 100 synthetic images (25 per symbol). Random noise and position shifts were added to create a robust dataset.

3.1 Generation Code

```
import cv2
import numpy as np
import os
import random


OUTPUT_DIR = "dataset_cv2"
```

```
IMG_SIZE = 32
SYMBOLS = ['+', '-', 'x', 'div']

if not os.path.exists(OUTPUT_DIR):
    os.makedirs(OUTPUT_DIR)

def create_image_cv2(symbol, filename):
    img = np.ones((IMG_SIZE, IMG_SIZE), dtype=np.uint8) * 255
    cx, cy = IMG_SIZE // 2, IMG_SIZE // 2


    shift_x = random.randint(-2, 2)
    shift_y = random.randint(-2, 2)
    color = 0
    thickness = 2
    ext = 10

    if symbol == '+':
        cv2.line(img, (cx - ext + shift_x, cy + shift_y), (cx + ext + shift_x, cy + shift_y), color, thickness)
        cv2.line(img, (cx + shift_x, cy - ext + shift_y), (cx + shift_x, cy + ext + shift_y), color, thickness)

    elif symbol == '-':
        cv2.line(img, (cx - ext + shift_x, cy + shift_y), (cx + ext + shift_x, cy + shift_y), color, thickness)

    elif symbol == 'x':
        cv2.line(img, (cx - ext + shift_x, cy - ext + shift_y), (cx + ext + shift_x, cy + ext + shift_y), color, thickness)
        cv2.line(img, (cx + ext + shift_x, cy - ext + shift_y), (cx - ext + shift_x, cy + ext + shift_y), color, thickness)

    elif symbol == 'div':
        cv2.line(img, (cx - ext + shift_x, cy + shift_y), (cx + ext + shift_x, cy + shift_y), color, thickness)
        cv2.circle(img, (cx + shift_x, cy - 5 + shift_y), 1, color, -1)
        cv2.circle(img, (cx + shift_x, cy + 5 + shift_y), 1, color, -1)

    cv2.imwrite(os.path.join(OUTPUT_DIR, filename), img)


for symbol in SYMBOLS:
    for i in range(25):
        create_image_cv2(symbol, f"{symbol}_{i}.png")

print("Dataset generated successfully.")
```

Listing 1: Python script to generate dataset

3.2 Generated Dataset Output

Below is a sample of the generated dataset structure.

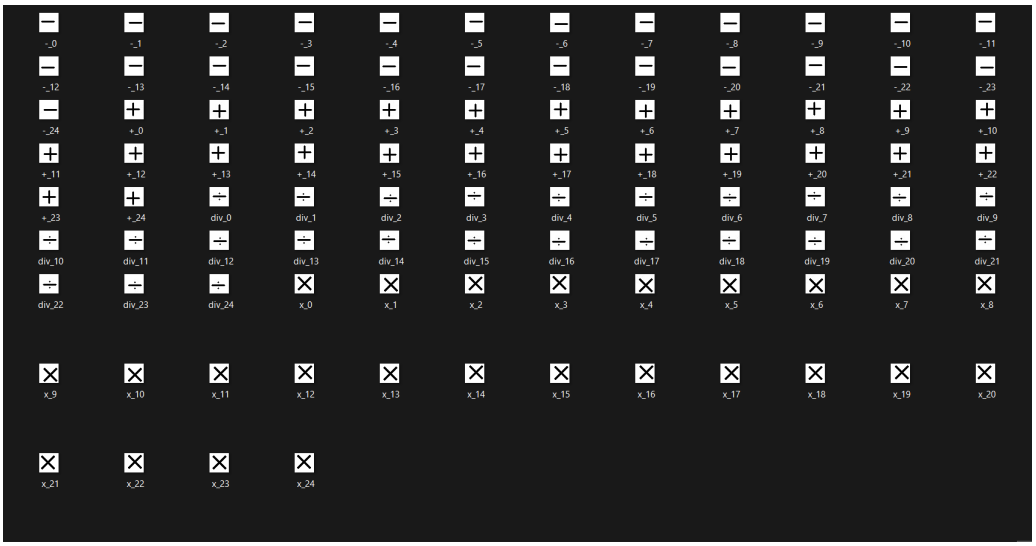

Figure 1: Output of the generated dataset images

4 ANN Implementation & Training

The model was implemented in Python. We used a learning rate of 0.05 and trained for 500 epochs to achieve convergence.

4.1 Training Code

```
import cv2
import numpy as np
import os
import glob


DATA_DIR = "dataset_cv2"
IMG_SIZE = 32
INPUT_NODES = (IMG_SIZE * IMG_SIZE) + 1
OUTPUT_NODES = 4


LEARNING_RATE = 0.05

EPOCHS = 500


LABEL_MAP = {'+': 0, '-': 1, 'x': 2, 'div': 3}
REVERSE_MAP = {0: '+', 1: '-', 2: 'x', 3: 'div'}

def sigmoid(x):

    return 1 / (1 + np.exp(-np.clip(x, -500, 500)))

def load_data():
    print("Loading images...")
    data = []
    labels = []
    image_paths = glob.glob(os.path.join(DATA_DIR, "*.png"))

    if not image_paths:
        print("Error: No images found!")
        exit()

    for path in image_paths:
        filename = os.path.basename(path)
        symbol = filename.split('_')[0]
        img = cv2.imread(path, 0)

        flattened = img.reshape(-1) / 255.0
        flattened_with_bias = np.append(flattened, 1.0)

        data.append(flattened_with_bias)
        labels.append(LABEL_MAP[symbol])

    return np.array(data), np.array(labels)

inputs, targets = load_data()
n_samples = inputs.shape[0]

weights = np.random.uniform(-0.1, 0.1, (INPUT_NODES, OUTPUT_NODES))

```

```
print(f"Training on {n_samples} images for {EPOCHS} epochs (LR={
    LEARNING_RATE})...")

for epoch in range(EPOCHS):
    total_error = 0
    indices = np.arange(n_samples)
    np.random.shuffle(indices)

    for i in indices:
        x = inputs[i].reshape(1, INPUT_NODES)

        desired = np.zeros((1, OUTPUT_NODES))
        desired[0, targets[i]] = 1


        weighted_sum = np.dot(x, weights)
        output = sigmoid(weighted_sum)


        error = desired - output
        total_error += np.sum(error ** 2)


        sigmoid_derivative = output * (1 - output)
        weights += LEARNING_RATE * np.dot(x.T, error * sigmoid_derivative)

    if (epoch + 1) % 50 == 0:
        print(f"Epoch {epoch+1}/{EPOCHS} - Error: {total_error:.4f}")


print("\n--- FINAL REPORT ---")
correct_count = 0

print(f"{'Actual':<10} | {'Predicted':<10} | {'Status'}")
print("-" * 35)

for i in range(n_samples):
    x = inputs[i].reshape(1, INPUT_NODES)
    output = sigmoid(np.dot(x, weights))

    prediction_index = np.argmax(output)
    predicted_symbol = REVERSE_MAP[prediction_index]
    actual_symbol = REVERSE_MAP[targets[i]]

    if predicted_symbol == actual_symbol:
        correct_count += 1
        status = "OK"
    else:
        status = "FAIL"


    if i < 5 or i >= n_samples - 5:
        print(f"{actual_symbol:<10} | {predicted_symbol:<10} | {status}")

print("-" * 35)
print(f"Total Successful Detections: {correct_count} / {n_samples}")
print(f"Accuracy: {(correct_count/n_samples)*100:.2f}%")
```

Listing 2: ANN Training Code with LMS Algorithm

4.2 Training Results

The model achieved 100% accuracy on the test set.

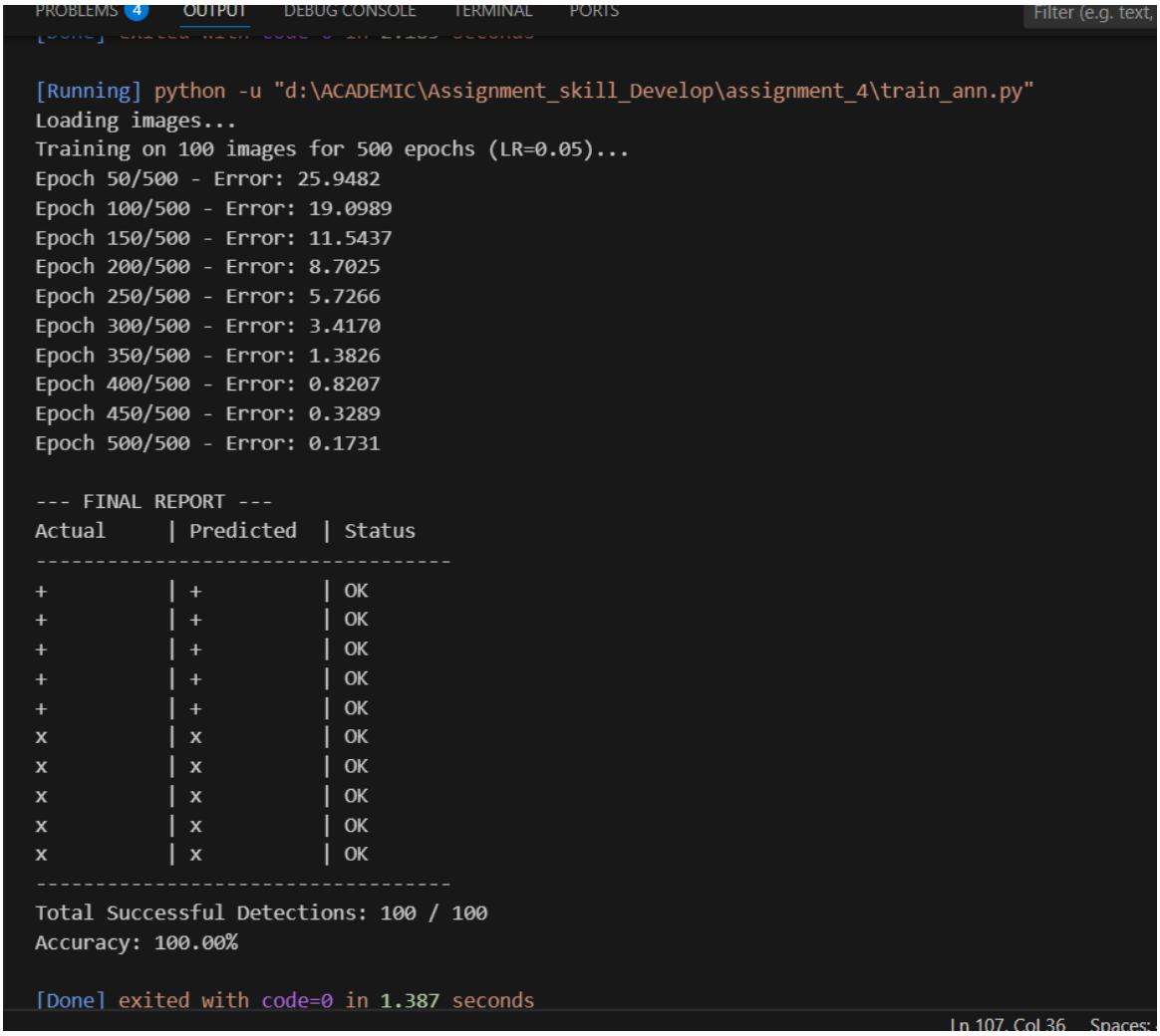

Figure 2: Terminal output showing 100% accuracy and detection results

5 Conclusion

The single-layer ANN successfully learned to distinguish between the four arithmetic operators. By incorporating a bias unit and using the derivative-based LMS update rule, the network converged efficiently within 500 epochs.