Green University of Bangladesh

Department of Computer Science and Engineering

Multilayer Perceptron

for Image Recognition

Course: **Machine Learning**

Submitted by:

Ibrahim Refat
ID: 221902333
Section: 221D9

Submitted to:

Dr. Muhammad Abul Hasan
Associate Chairperson

September 1, 2025

Contents

Abstract

This report details a project focused on creating an animated visualization of a Multilayer Perceptron (MLP) performing a basic image recognition task. The objective was to demonstrate the fundamental principles of how a neural network learns to identify a specific pattern—in this case, the letter 'L' on a 5x5 pixel grid. Using the Python library Manim, an animation was developed to illustrate the core processes of data preparation, forward propagation, and the learning mechanism of backpropagation. This project serves as a practical, visual introduction to the inner workings of MLPs, translating abstract concepts into an intuitive and understandable format.

Chapter 1

Introduction

A Multilayer Perceptron (MLP) is a foundational class of feedforward artificial neural network, drawing its inspiration from the structure of biological neurons. It consists of at least three layers of nodes: an input layer, one or more hidden layers, and an output layer. Each neuron in one layer is fully connected to the neurons in the subsequent layer, allowing information to flow in a single direction. Through a process of learning from data, MLPs can identify complex patterns and are widely used in various applications, including image recognition, speech processing, and classification tasks.

The goal of this project was to demystify the operations of an MLP by creating an animated video that visually explains, step-by-step, how it can be trained to recognize a simple image.

Chapter 2

Project Objective

The primary objective of this project was to develop an animated visualization that illustrates how a Multilayer Perceptron learns to perform image recognition.

To achieve this, a simple and clear use case was defined:

- **Task:** Detect the letter 'L' within a 5x5 pixel grid.
- **Input Data:** A 5x5 grid where each pixel is either "on" (100% opacity) or "off" (0% opacity). The letter 'L' is formed by 7 activated pixels.
- **Visualization Tool:** The Manim (Mathematical Animation Engine) library in Python.

The animation aims to guide the viewer through the entire process, from converting the image into a format the network can understand, to the final recognition result after training.

Chapter 3

Methodology

The process of training an MLP to recognize the letter 'L' can be broken down into four key steps, which are visualized in the animation and detailed in the figures in Chapter 5.

3.1 Step 1: Data Preparation - Image to Vector

The neural network cannot process an image in its 2D grid format directly. The first step is to convert the 5x5 pixel grid into a one-dimensional list, or a "flattened vector" (see Figure 5.1). Each pixel from the grid is taken in order (e.g., row by row) and placed into a single line of 25 values. An "on" pixel is represented by a 1, and an "off" pixel by a 0. This vector of 25 numbers serves as the input for our network.

3.2 Step 2: Network Architecture

The MLP for this task was designed with a standard feedforward structure (see Figure 5.2).

- **Input Layer:** Contains 25 neurons, one for each pixel in the input image. A neuron is activated if its corresponding pixel value is 1.
- **Hidden Layers:** One or more intermediate layers that perform the critical computations. These layers take the inputs, transform them, and pass them forward, allowing the network to learn complex relationships between the input pixels.
- **Output Layer:** The final layer that produces the prediction. For this classification task (is it an 'L' or not?), it could be a single neuron that outputs a value indicating the probability that the image is an 'L'.

3.3 Step 3: Forward Propagation

Once the input vector is ready, it is fed into the network in a process called **forward propagation** (visualized in Figure 5.3).

1. **Linear Combination:** Each neuron in the hidden layer calculates a weighted sum of the outputs from the input layer. It adds all the input values multiplied by their corresponding connection weights, plus a bias term.
2. **Activation Function:** This sum is then passed through a non-linear **activation function** (like Sigmoid or ReLU). This function decides whether the neuron should be "activated" or not, introducing the ability to learn complex patterns.
3. This process repeats for each layer until a final value is produced by the output layer.

3.4 Step 4: Training with Backpropagation

Initially, the network's weights and biases are random, so its first prediction will be incorrect. The network learns by correcting its mistakes through a process called **backpropagation** (visualized in Figure 5.4).

1. **Error Calculation:** The network's output is compared to the correct label (e.g., "1" for 'L', "0" for not 'L'). The difference between the predicted output and the actual label is the **error**.

2. **Propagate Error Backwards:** This error is propagated backward through the network, from the output layer to the input layer.

3. **Update Weights:** The network uses the error to calculate how much each weight and bias contributed to the mistake. It then adjusts these parameters slightly in the direction that will reduce the error in the next attempt.

This cycle is repeated many times (over many "epochs") until the network's weights are fine-tuned to accurately recognize the letter 'L'.

Chapter 4

Implementation

The complete source code for the Manim animation can be found on GitHub.

```
https://github.com/ibrahimrifats/ML-with-Manim/blob/main/MLP.py
```

Listing 4.1: GitHub Repository URL

The project can be accessed via the hyperlink: https://github.com/ibrahimrifats/ML-with-Manim

Chapter 5

Figures

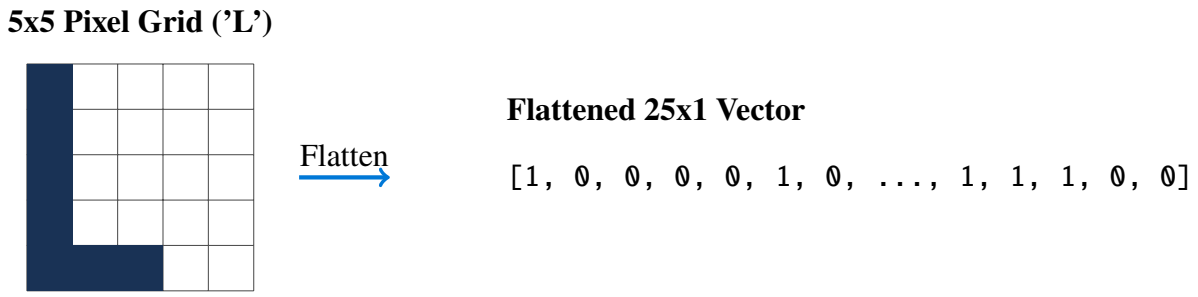

Figure 5.1: Converting the 2D pixel grid of the letter 'L' into a 1D flattened vector for the input layer.

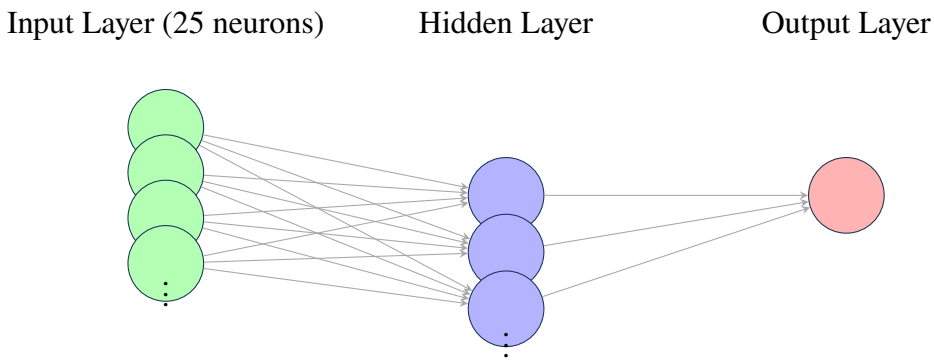

Figure 5.2: Architecture of the Multilayer Perceptron with an input, hidden, and output layer.

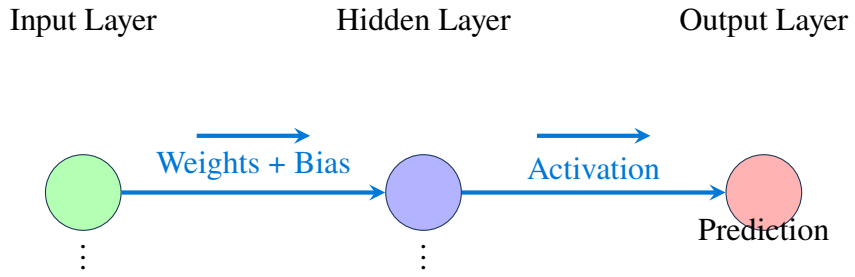

Figure 5.3: Illustration of Forward Propagation, where data flows from input to output.

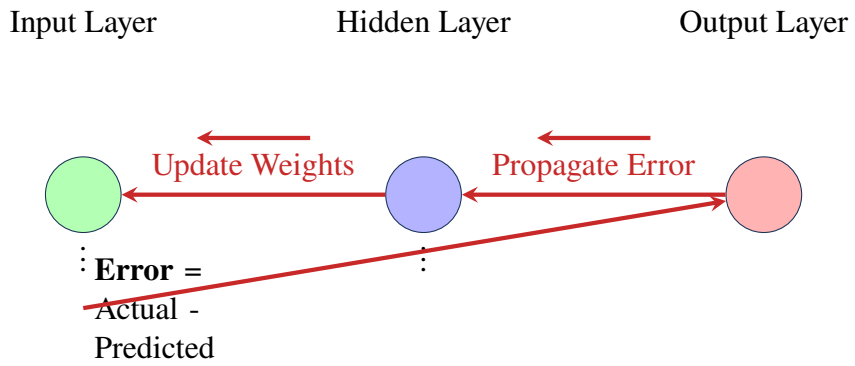

Figure 5.4: Illustration of Backpropagation, where the error is propagated backward to update weights.

Chapter 6

Challenges

The main challenge in this project was learning the Manim library. Making accurate and well-timed animations took a lot of effort, especially with limited time.

Chapter 7

Conclusion

This project simplifies complex neural network concepts like data vectorization, forward propagation, and backpropagation using an animated demonstration of a Multilayer Perceptron recognizing the letter 'L'. Despite video quality limitations, the project successfully makes these abstract ideas tangible and understandable.

References

1. Manim Community Documentation. (2023). Retrieved from https://docs.manim.community/en/stable/

2. GeeksforGeeks. (n.d.). *Multi-Layer Perceptron Learning in TensorFlow*. Retrieved from https://www.geeksforgeeks.org/deep-learning/multi-layer-perceptron-learning-in

3. DataCamp. (n.d.). *Multilayer Perceptrons in Machine Learning*. Retrieved from https://www.datacamp.com/tutorial/multilayer-perceptrons-in-machine-learning