Hand-written Digit Recognition using Numpy

Yashwant Bhosale

January 2025

Contents

1 Overview

This project demonstrates working of a simple neural network using the MNIST dataset. The special thing about this project is that it is implemented only using numpy and no other libraries. The neural network is trained using the *backpropagation algorithm*. The neural network is trained on the MNIST dataset and is able to achieve an accuracy of 97.8% on the test dataset. The project also contains a simple web interface to draw digits and predict the digit using the trained neural network.

1.1 Dataset

The neural network is trained on **MNIST Dataset**. It is set of about 70,000 images divided into training data and testing data. Training data contains 60,000 images and testing data contains 10,000 images.

1.2 Technologies used

The approach of this project was to learn how neural networks work by implementing them from scratch. So, the neural network is written in python and only using NumPy which is popular library in python used to deal with numbers.

1.3 Approach

The appoach is layered-neural network. The neural network has

1. **Input layer:** It has 784 neurons. Each image in dataset is 28 pixels wide and 28 pixels long. This layer is constructed by flattening array of RGBA values of pixels of a 28x28 image.
2. **Hidden layers:** My network contains two hidden layers one having 128 neurons and other having 64 neurons. These layers play important role of capturing features of each image.
3. **Output layer:** Output layer contains 10 neurons representing 10 digits. Network will try to confidently classify image as one of the 10 digits.

2 Parts of Neural Network

There are various elements that constitute a neural network. It is essential to know what is significance of each of them:

- **Neuron**: Neuron can be looked at as a simple container that holds some value. There are layers in the network and each layer contains some neurons. Activation of neurons in one layer defines the activations in the next and so on.

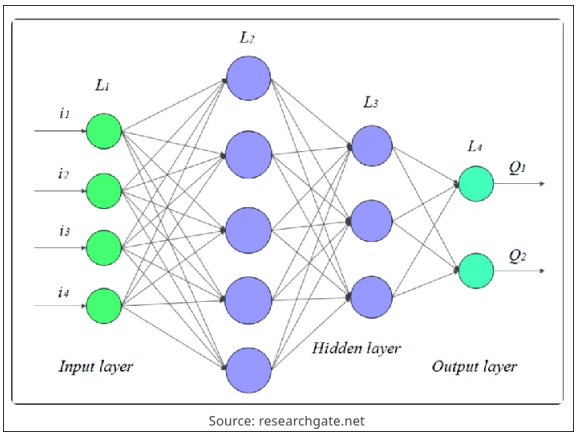


Figure 1: Example image demonstrating neural networks

- **layers**: There are layers in the neural network and it can be seen in the image. You may look at layers as something which is expected to record some specific feature of the image.
- **Weights and Biases**: Weights and biases are parameters in neural networks that control the strength of connections between neurons and the output of the network. They are learned during the training process and are adjusted to optimize the network's performance
- **Cost function**: A cost function in a neural network is a measure of how far the predicted output is from the actual output. It's also known as a loss function. We essentially look at how wrong are we.
- **Activation function**: An activation function is a mathematical function applied to the output of a neuron. It introduces non-linearity into the model, allowing the network to learn and represent complex patterns in the data.

$$a^L = w^{L-1} \cdot a^{L-1} + b^{L-1}$$

a^L: Neuron activations in L^{th} layer
w^{L-1}: weights of $(L-1)^{th}$ layer
a^{L-1}: Neuron activations in $(L-1)^{th}$ layer
b^{L-1}: Biases of $(L-1)^{th}$ layer

This function sort of **governs** the neural networks. You may understand this as we discussed earlier activations, weights and biases of one layer determine the activation in the next layer.
Examples of cost functions:

- **Mean Square Error**:

$$Cost = \sum_{i=1}^{n} (y - y_i)^2$$

- **Cross Entropy Loss**

$$Cost = -(y \log(p) + (1 - y) \log(1 - p))$$

3 Technical concepts

This section demonstrates the usage of advanced LaTeX features including equations, hyperlinks, tables, lists, and diagrams.

3.1 Mathematical Equations

Here is an example of a mathematical equation used for weight updates in backpropagation:

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \eta \frac{\partial C}{\partial w_{ij}} \tag{1}$$

$$b_j^{(t+1)} = b_j^{(t)} - \eta \frac{\partial C}{\partial b_j}, \tag{2}$$

where:

- w_{ij}: Weight between neuron i in layer L and neuron j in layer $L + 1$
- η: Learning rate
- $\frac{\partial C}{\partial w_{ij}}$: Gradient of cost function

3.2 Dataset

For more information on the MNIST dataset, visit the official MNIST Database website.

3.3 Different activation function

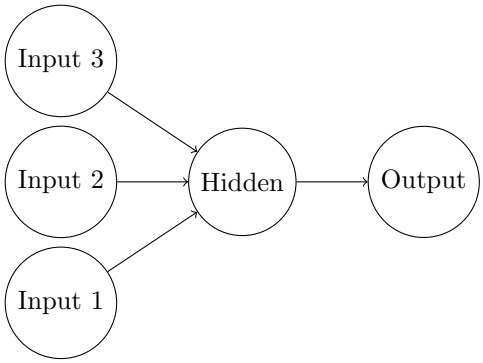
The following table illustrates a comparison of activation functions:

u8i

Activation Function	**Formula**	**Range**
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	(0, 1)
ReLU	$f(x) = \max(0, x)$	[0, ∞]
Tanh	$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	(-1, 1)

3.4 High level diagram

Below is a simple diagram describing neural network



3.5 Code Blocks

The following is an example Python code snippet demonstrating weight initialization:

```
import numpy as np
weights = np.random.randn(784, 128) * 0.01
biases = np.zeros((128, 1))
```

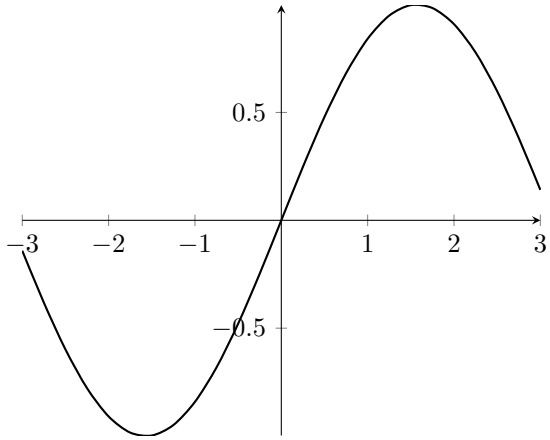
Listing 1: Weight Initialization

3.6 How to train a neural network

Steps for training a neural network:

1. Initialize weights and biases.
2. Feedforward to compute outputs.
3. Compute the cost function.
4. Backpropagate to update weights.
5. Repeat until convergence.

4 Testing



The equation of straight line in intercept form is $y = mx + c$

Data Table with Links

Here is a table where each row links to multiple external files.

Data Name	File 1	File 2
Experiment A	Data 1	Data 2
Experiment B	Report	Raw Data
Experiment C	Dataset C	Notes

Table 1: Table with links to multiple external files