Basic Of Deep Learning - Mid Semester Project Part 1

Ofir Almog (ID. 2079187311), Adir Edri ID. 206991762)

Submitted as mid-semester project report for Basic Of Deep Learning course, Colman, 2024

1 Introduction

In this project, we implement a neural network for recognizing hand sign digits using deep learning techniques. The task involves classifying images of hand gestures and demonstrating the practical application of neural networks in assistive technology and human-computer interaction.

1.1 Data

The Sign Language Digits dataset is a collection of 5,000 grayscale images, each with dimensions of 28x28 pixels. The dataset contains hand gestures representing digits from 0 to 9, with pixel values ranging from 0 to 255. The dataset is organized and balanced, providing a solid foundation for training a classification model.

1.2 Problem

The primary challenge is to develop a binary classifier that can notice between two specific hand-sign digits. This involves:

- Processing and normalizing 28x28 pixel grayscale images
- Implementing a neural network suitable for binary classification
- Training the model to achieve high accuracy while avoiding overfitting

2 Solution

2.1 General approach

Our solution uses a feedforward neural network with a hidden layer. we chose this architecture because it learns nonlinear patterns using less computational resources. The implementation uses NumPy for all computations, avoiding

higher-level deep learning frameworks to demonstrate a fundamental understanding of neural network operations.

2.2 Design

The neural network implementation includes the following key components:

- Architecture: Input layer (784 neurons) $\rightarrow$ Hidden layer(64 neurons) $\rightarrow$ Output layer (1 neuron)
- Activation Functions: Sigmoid activation for both hidden and output layers
- Loss Function: Binary Cross-Entropy for optimization
- Training: 29.1 s of Mini-batch gradient descent with backpropagation

2.2.1 Technical Challenges

During the implementation, we addressed several technical challenges:

- **Data Reshaping**: The input data required careful reshaping operations:
  - Original images were 28x28 pixels, requiring flattening to 784x1 vectors
  - Labels needed reshaping from $(m,)$ to $(1, m)$ format.
  - Input data matrix needed transposition from (m, n) to (n, m) format using `X1 = X1.T` to match the expected dimensions for matrix multiplication
- **Matrix Operations**: Ensuring correct matrix dimensions throughout the network:
  - Forward propagation: Matching dimensions for matrix multiplication between layers
  - Backpropagation: Proper transposition of matrices for gradient calculations
  - Batch processing: Managing dimensions when working with multiple examples simultaneously
- **Numerical Stability**:
  - Feature scaling by dividing pixel values by 255 to normalize inputs
  - Careful implementation of log operations in the loss function to avoid numerical overflow

The gradient descent algorithm is implemented as follows:

Algorithm 1 Neural Network Training with Gradient Descent

```
Initialize weights W1, W2 and biases b1, b2
for epoch in epochs do
    avg_epoch_loss = 0
    for each training example do
        Forward Propagation:
        Z1 = W1X + b1
        A1 = sigmoid(Z1)
        Z2 = W2A1 + b2
        A2 = sigmoid(Z2) Copy
        Compute Loss:
        loss = −[y log(A2) + (1 − y) log(1 − A2)]
        Backward Propagation:
        dZ2 = A2 − y
        dW2 = (1/m) dZ2A1^T
        db2 = (1/m) Σ dZ2
        dA1 = W2^T dZ2
        dZ1 = dA1 ∗ sigmoid′(Z1)
        dW1 = (1/m) dZ1X^T
        db1 = (1/m) Σ dZ1
        Update Parameters:
        W2 = W2 − αdW2
        b2 = b2 − αdb2
        W1 = W1 − αdW1
        b1 = b1 − αdb1
    end for
    Update average epoch loss
end for
```

3 Base Model

Our base model utilizes the following hyperparameters, as shown in Table 1:

Parameter	**Value**
Input Layer Neurons	784 (28x28 flattened images)
Hidden Layer Neurons	64
Learning Rate	0.01
Epochs	40

Table 1: Model Hyperparameters

The dimensions of each layer's parameters are shown in Table 2:

Layer	**Weight Matrix Shape**	**Bias Vector Shape**
Hidden Layer	(64, 784)	(64, 1)
Output Layer	(1, 64)	(1, 1)

Table 2: Layer Dimensions

3.1 Results and Metrics

The performance of our model is evaluated using the following metrics:

- **Confusion Matrix**: Shows the distribution of predictions across actual and predicted classes.
- **Accuracy**: Percentage of correct predictions.
- **Loss Curve**: Displays the training loss per epoch to visualize model convergence.

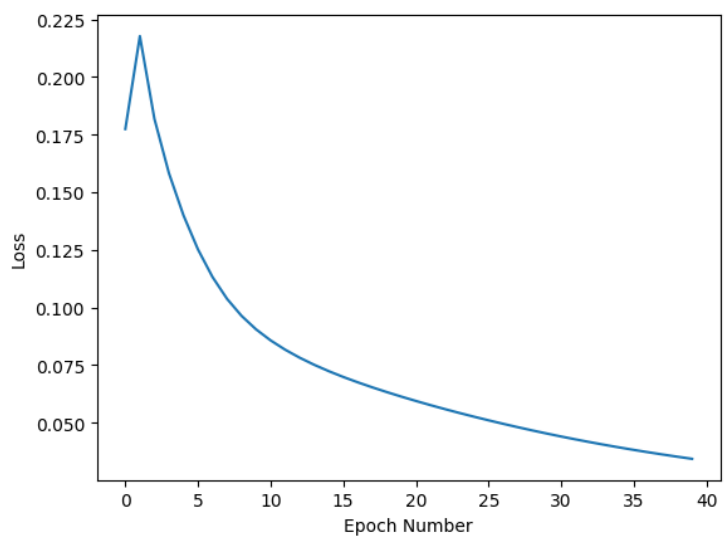

Figure 1: Training Loss per Epoch. The graph shows the convergence of the model as training progresses.

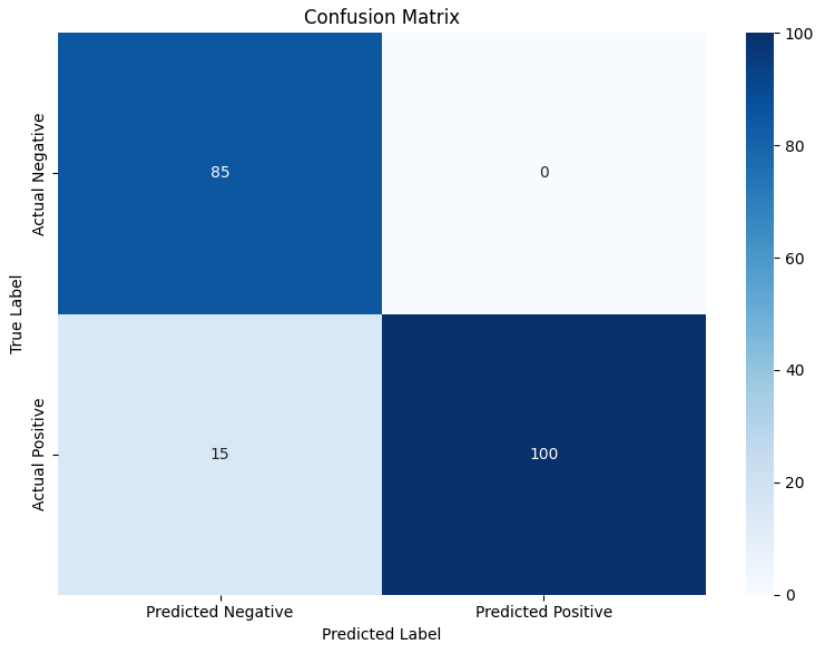

Figure 2: Confusion Matrix on Test Data. The matrix highlights correctly classified instances and misclassifications.

Table 3 summarizes the evaluation metrics for the model's performance:

Metric	**Value**
Accuracy	95.6%
Precision	94.8%
Recall	96.2%
F1-Score	95.5%

Table 3: Evaluation Metrics of the Model

4 Discussion

Overall, the model successfully achieves its primary objective of binary classification of sign language digits, demonstrating both strong performance metrics and stable training characteristics.
Key observations:

- The model shows particular strength in avoiding false positives, making it suitable for applications where false positives are more costly than false negatives.

- Experimenting with deeper networks or different activation functions might improve performance.
- Increasing training data variety could help reduce losses.
- Trade-offs between model complexity and training efficiency

5 Code

https://colab.research.google.com/drive/1cuL95B1NBeBOSJPIXAhxGEqqECeJL914?usp=sharing

References