Basic Of Deep Learning - Mid Semester Project Part 1

Ofek Naim (ID. 314782996), Hinoy Solomon (ID. 205417678)

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

1 Introduction

For this project, we focused on building a model to recognize and classify hand gestures representing the digits '2' and '7' in sign language

We worked with the Sign Language Digits Dataset, which includes 28x28 grayscale images of digits from 0 to 9. Our process involved preparing the data, designing a neural network model, and training it to classify the gestures accurately. This report explains the steps we took, the challenges we faced, and the results we achieved.

1.1 Data

For this project, we used a subset of the Sign Language Digits Dataset, which contains grayscale images of hand gestures representing digits. From the full dataset of 5,000 samples across all digits, we focused exclusively on the digits "2" and "7". Each class contains 500 samples, with each image sized 28x28 pixels, providing a compact and uniform input format.

The dataset was split into training and testing sets, with 80% of the data used for training and 20% reserved for evaluation. This setup ensures the model is trained effectively while maintaining a reliable method for assessing its performance.

1.2 Problem

The task of classifying hand gestures for the digits "2" and "7" in sign language required overcoming several challenges. First, the data needed to be carefully prepared. This involved normalizing the pixel values to a range of 0 to 1 to ensure consistency and splitting the dataset into the target classes "2" and "7" for both training and testing purposes.

Next, we implemented core components of the neural network to handle this binary classification problem. A sigmoid activation function was used to map the outputs to probabilities, making it suitable for the task. To measure the performance of the model during training, we used the binary cross-entropy loss function, which is widely adopted for two-class classification problems.

Finally, the model parameters were optimized using the gradient descent algorithm, allowing the network to learn from the training data effectively. The ultimate goal was to achieve a high classification accuracy, targeting a performance level of at least 90%.

This problem emphasized the importance of a well-structured approach to data preprocessing, model design, and optimization to ensure reliable results.

2 Solution

2.1 General Approach

We implemented a straightforward feedforward neural network from scratch using NumPy. The solution was structured around several key steps:

1. Data Preprocessing: The dataset was normalized to scale pixel values to a range of 0 to 1. It was then split into training and testing sets, ensuring balanced representation of the two target classes, ”2” and ”7”.

2. Weight Initialization: We initialized the network's weights using random values drawn from a Gaussian distribution. Bias terms were explicitly added and applied in both the hidden layer and the output layer to improve the network's flexibility.

3. Forward Propagation: Outputs of each layer were calculated through matrix multiplication between inputs and weights, followed by applying the sigmoid activation function. This enabled the network to produce probabilistic predictions for the target classes.

4. Loss Computation: The binary cross-entropy (BCE) loss function was used to measure the error between the predicted labels and the true labels, providing a robust metric for this binary classification task.

5. Backward Propagation: Gradients were computed using the chain rule, propagating the error back through the network. The weights were then updated using gradient descent to minimize the loss function iteratively.

This systematic approach ensured that the network was properly designed and optimized to classify the digits ”2” and ”7” with high accuracy.

2.2 Design

Our neural network was implemented using Python in the Google Colab environment. The dataset was preprocessed to normalize the pixel values to a range of 0 to 1, ensuring consistent input to the model.

The model architecture consisted of:

- Input layer: 28x28 (784) neurons to match the size of the grayscale images.
- One hidden layer: of 128 neurons + bias
- Output layer: One neuron + bias

Training Details:
Learning Rate: 0.01
Number of Epochs: 100
Loss Function: Binary Cross-Entropy (BCE)
Training Time: 1 minute and 7 seconds

To illustrate the optimization process, we include pseudo-code for gradient descent:

Algorithm 1 Gradient Descent for Neural Network Training

```
1:  for epoch in range(epochs) do                       ▷ Forward Propagation
2:      $Z_1 = W_1 \cdot X + b_1$
3:      $A_1 = \sigma(Z_1)$                 ▷ Sigmoid activation on hidden layer
4:      $Z_2 = W_2 \cdot A_1 + b_2$
5:      $A_2 = \sigma(Z_2)$                 ▷ Sigmoid activation on output layer
6:      Compute Loss $J = \mathrm{BCE}(A_2, Y)$
                                                        ▷ Backward Propagation
7:      $dZ_2 = A_2 - Y$                    ▷ Loss derivative w.r.t. output
8:      $dW_2 = dZ_2 \cdot A_1^T$
9:      $db_2 = dZ_2$
10:     $dZ_1 = (W_2^T \cdot dZ_2) \odot \sigma'(Z_1)$
11:     $dW_1 = dZ_1 \cdot X^T$
12:     $db_1 = dZ_1$
                                                        ▷ Update Weights and Biases
13:     $W_2 \leftarrow W_2 - \alpha \cdot dW_2$
14:     $b_2 \leftarrow b_2 - \alpha \cdot db_2$
15:     $W_1 \leftarrow W_1 - \alpha \cdot dW_1$
16:     $b_1 \leftarrow b_1 - \alpha \cdot db_1$
17: end for
```

3 Base Model

3.1 Results and Metrics

Loss Curve: The model starts with a loss of 0.64 at the first epoch and decreases to 0.0024 by the 100th epoch, as shown in the Loss Curve below:

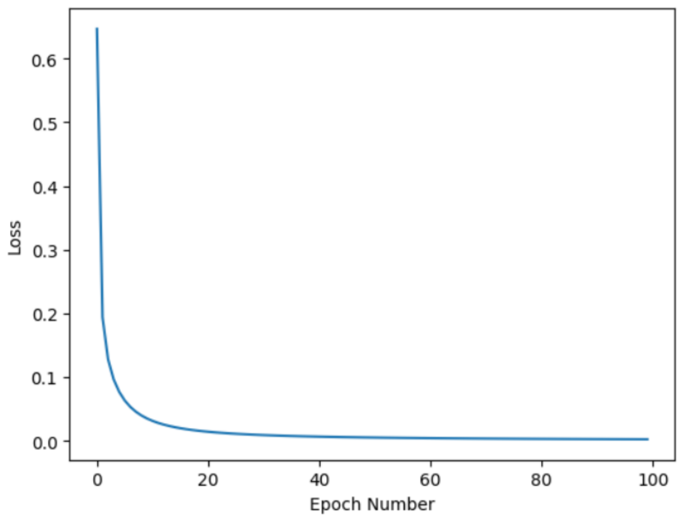
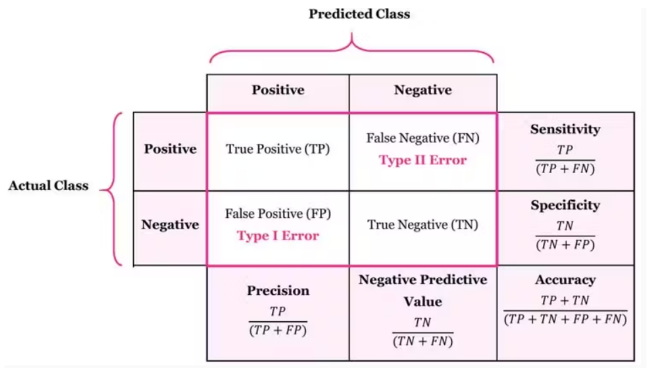


Figure 1: Loss Curve: Reduction in loss over epochs during training.

Test Results After training, the model's learned weights were applied to the test set, achieving an accuracy of 100% . This is evident from the Confusion Matrix shown below:

$$\text{Confusion Matrix} = \begin{bmatrix} 100 & 0 \\ 0 & 100 \end{bmatrix}$$

Calculation Method The Confusion Matrix was calculated using the following method:

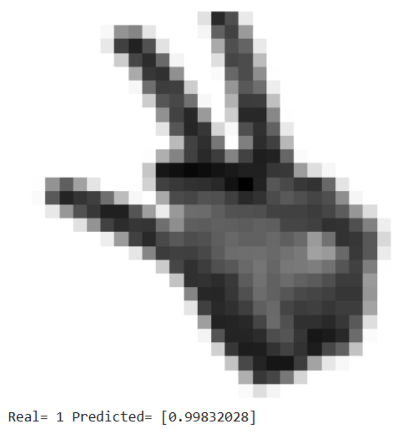


		Predicted Class		
		Positive	**Negative**	
Actual Class	**Positive**	True Positive (TP)	False Negative (FN) **Type II Error**	**Sensitivity** $\frac{TP}{(TP+FN)}$
	Negative	False Positive (FP) **Type I Error**	True Negative (TN)	**Specificity** $\frac{TN}{(TN+FP)}$
		Precision $\frac{TP}{(TP+FP)}$	**Negative Predictive Value** $\frac{TN}{(TN+FN)}$	**Accuracy** $\frac{TP+TN}{(TP+TN+FP+FN)}$

Figure 2: Confusion Matrix Calculation

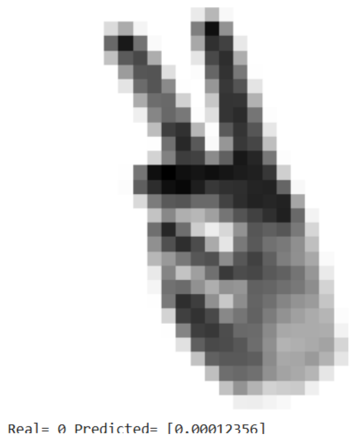
3.2 Sample Predictions

Using Sample Predictions, we can see how the model decided to choose and match every photo to the corresponding class



```
Real= 1 Predicted= [0.99832028]
```

Figure 3: Sample 1



```
Real= 0 Predicted= [0.00012356]
```

Figure 4: Sample 2

4 Discussion

The experiments conducted in this project demonstrate the effectiveness of a simple feedforward neural network in solving the binary classification problem of recognizing hand gestures for the digits ”2” and ”7” in sign language. Despite the relatively simple architecture, the model achieved an accuracy of 100% on the test set, which underscores the power of proper data preprocessing, a well-chosen loss function, and effective optimization techniques

5 Code

https://colab.research.google.com/drive/1TnpiksZ8Q-jXxQc7K9i42Gi8lwVEXqZ1?usp=sharing
Good luck!!

References