Basic Of Deep Learning - Mid Semester Project Part 1

Shir Zohar (ID. 323856542), Sivan Zagdon (ID. 213002918)

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

1 Introduction

Deep learning is a subset of machine learning that uses neural networks with multiple layers to identify patterns and relationships in data. Its applications span various fields, such as image recognition and natural language processing. The goal of this project is to classify hand-written digits in sign language using a neural network built from scratch with NumPy.

This project focuses on implementing key deep learning techniques like forward and backward propagation, gradient descent, and parameter optimization. By avoiding high-level libraries like Keras and PyTorch, the implementation highlights the mathematical foundations of deep learning and provides a deeper understanding of its core principles.

The strength of deep learning lies in its ability to automatically extract features and model non-linear relationships, making it highly effective for tasks like digit classification. Building the network from scratch ensures a practical grasp of these concepts and their application to real-world challenges.

1.1 Data

The dataset consists of grayscale images of hand-written digits (0 to 9), resized to 28x28 pixels. For this project, we focus on binary classification between digits 4 and 5. Preprocessing steps include:

- Normalizing pixel values to the range [0, 1].
- Reshaping data to match the neural network's input format.
- Splitting data into training (80%) and testing (20%) sets.

The dataset is imbalanced, requiring careful preprocessing to ensure fair representation of the selected classes.

1.2 Problem

The project addresses the binary classification of hand-written digits into two categories: 4 and 5. This task poses several challenges:

- Variability in handwriting styles, introducing noise into the dataset.
- Ensuring generalization to unseen data while avoiding overfitting.
- Implementing the neural network from scratch using only NumPy.
- Optimizing weights and biases through iterative training to minimize loss.
- Addressing class imbalance to ensure balanced learning.

This project demonstrates the applicability of deep learning techniques to small-scale tasks and provides practical experience in building and tuning a neural network from scratch. It emphasizes the importance of reproducible and well-documented experiments, paving the way for tackling more complex problems in the future.

2 Solution

2.1 General Approach

To solve the binary classification problem for hand-written digits, we adopt a systematic approach by building a neural network from scratch using only NumPy. The key steps include:

1. **Data preprocessing:** Normalize pixel values to [0, 1], reshape data to match the input layer, and split it into training and testing sets.
2. **Model architecture:** Design a network with 784 input neurons (28x28 pixels), a hidden layer with 32 neurons, and a single output neuron for binary classification.
3. **Forward propagation:** Compute activations using the sigmoid function to introduce non-linearity.
4. **Loss function:** Use binary cross-entropy to measure prediction accuracy.
5. **Backward propagation:** Calculate gradients and update weights and biases using gradient descent.
6. **Training:** Train the model over multiple epochs while monitoring the loss for convergence.
7. **Evaluation:** Assess model performance on unseen data using accuracy and confusion matrices.

This approach effectively addresses the classification problem while adhering to the principles of reproducibility and interpretability in machine learning.

2.2 Design

Platform: The code was implemented in Python using NumPy and executed in Google Colab for efficient computational resources.

Training Time: Training required approximately 10 minutes for 50 epochs on the dataset.

Challenges:

- Debugging backpropagation to ensure accurate gradient calculations.
- Addressing data imbalance through preprocessing and shuffling.
- Avoiding overfitting by tuning the learning rate.

Loss Function: Binary cross-entropy, ideal for binary classification, was used to minimize prediction errors.

Activation Function: The sigmoid function introduced non-linearity and mapped outputs to probabilities.

Architecture: The neural network included:

- An input layer with 784 neurons (28x28 pixel images).
- A hidden layer with 32 neurons using sigmoid activation.
- An output layer with 1 neuron for binary classification.

Algorithm 1 Gradient Descent Algorithm for Two-Layer Neural Network

Require: Learning rate α, number of epochs n, training data (X, Y)
Ensure: Optimized weights W_1, W_2, biases b_1, b_2
  Initialize W_1, b_1, W_2, b_2 with small random values
  for *epoch* $= 1$ to n **do**
    for each example (x_j, y_j) in (X, Y) **do**
      Forward Propagation:
      $Z_1 \leftarrow W_1 x_j + b_1$
      $A_1 \leftarrow \sigma(Z_1)$
      $Z_2 \leftarrow W_2 A_1 + b_2$
      $A_2 \leftarrow \sigma(Z_2)$
      Compute Loss:
      $\mathcal{L} \leftarrow -y_j \log(A_2) - (1 - y_j) \log(1 - A_2)$
      Backward Propagation:
      $dZ_2 \leftarrow A_2 - y_j$
      $dW_2 \leftarrow dZ_2 A_1^\top$
      $db_2 \leftarrow dZ_2$
      $dA_1 \leftarrow W_2^\top dZ_2$
      $dZ_1 \leftarrow dA_1 \cdot \sigma'(Z_1)$
      $dW_1 \leftarrow dZ_1 x_j^\top$
      $db_1 \leftarrow dZ_1$
      Update Parameters:
      $W_2 \leftarrow W_2 - \alpha dW_2$
      $b_2 \leftarrow b_2 - \alpha db_2$
      $W_1 \leftarrow W_1 - \alpha dW_1$
      $b_1 \leftarrow b_1 - \alpha db_1$
    end for
  end for
    return W_1, b_1, W_2, b_2

3 Base Model

The **base model** serves as the foundation for this project, built from scratch using only the NumPy library without relying on high-level machine learning libraries such as Keras or PyTorch. Below is a detailed explanation of the model's design, architecture, and implementation.

3.1 Model Architecture

The base model is a simple neural network designed for binary classification. It includes the following components:

- **Input Layer:**
  - The input layer consists of 784 neurons, corresponding to the 28x28 pixel grayscale images flattened into a vector.

- **Hidden Layer:**
  - Contains 32 neurons.
  - **Activation Function:** Sigmoid, which introduces non-linearity and maps values to the range [0, 1].
- **Output Layer:**
  - A single neuron is used for binary classification.
  - **Activation Function:** Sigmoid, converting the output to a probability.

3.2 Training Setup

- **Loss Function:** Binary Cross-Entropy Loss, ideal for binary classification tasks. It penalizes incorrect predictions proportionally to their confidence.
- **Optimizer:** Gradient Descent, used for optimizing the weights and biases of the model.
  - **Learning Rate:** 0.01, chosen to balance speed and stability of convergence.
- **Epochs:** The model is trained over 50 epochs to allow sufficient learning while preventing overfitting.
- **Data Preprocessing:**
  - The input data is normalized to the range [0, 1] to improve training stability.
  - Data is split into 80% training data and 20% testing data.

3.3 Model Implementation

The neural network was implemented from scratch using NumPy. The key components include:

- **Forward Propagation:** Computes the intermediate activations and predictions based on the input data.
- **Backward Propagation:** Calculates the gradients of the loss function with respect to weights and biases, updating the parameters using gradient descent.

3.4 Training Process

- **Initialization:** Weights and biases are initialized with small random values to prevent symmetry.
- **Convergence:** The training process shows a clear reduction in loss over 50 epochs, as evidenced in the training loss graph (see Section 1).

3.5 Results and Metrics

3.5.1 Training Loss

The graph below illustrates the training loss over 50 epochs:

- **X-axis (Epochs):** Represents the number of training cycles.
- **Y-axis (Loss):** Represents the error of the model during training.

The graph shows:

- A **steep decline** in the initial epochs, indicating rapid learning as the model adjusts its parameters effectively.
- A **gradual flattening** of the curve in later epochs, suggesting that the model converges to a minimal loss value, demonstrating successful optimization.

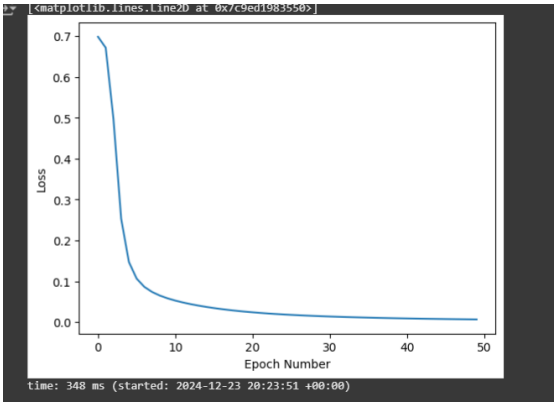

Figure 1: Training Loss over 50 Epochs

3.5.2 Confusion Matrix and Accuracy

The confusion matrix below shows the performance of the model on the test set:

True Label \Predicted Label	**0**	**1**
0	101	2
1	3	94

Table 1: Confusion Matrix of Model Predictions

- **True Positives (TP):** 94 instances where the model correctly predicted class 1.

- **True Negatives (TN):** 101 instances where the model correctly predicted class 0.
- **False Positives (FP):** 2 instances where the model incorrectly predicted class 1.
- **False Negatives (FN):** 3 instances where the model incorrectly predicted class 0.

Accuracy: 97.5%

```
Confusion Matrix:
 [[101   2]
 [  3  94]]
Accuracy: 0.975
```

This high accuracy indicates that the model is highly effective at distinguishing between the two classes.

3.5.3 Prediction Example

The image below shows a specific example of a model prediction:

- **Real Label:** 1
- **Predicted Label:** 1
- **Predicted Probability:** 0.9996

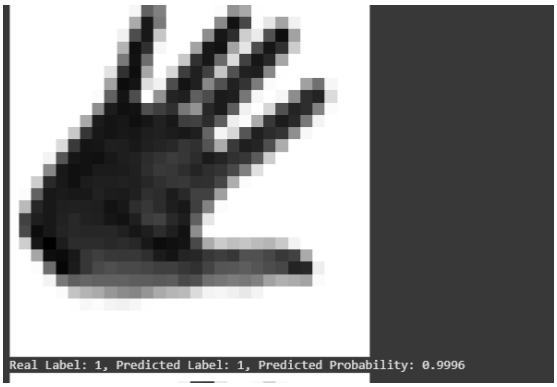

Figure 2: Example Model Prediction with High Confidence

3.5.4 Summary

- The **training loss graph** demonstrates effective learning and convergence.
- The **confusion matrix** confirms that the model achieves high accuracy with minimal false predictions.
- The **prediction example** showcases the model's ability to make confident and correct classifications.

These results indicate that the model is well-trained and optimized for the task, with excellent generalization capabilities. Future improvements could include testing on a larger dataset or exploring additional metrics like Precision, Recall, and F1-Score to further validate the model's performance.

4 Discussion

The experiments conducted in this project demonstrate the effectiveness of a simple neural network, implemented from scratch using only NumPy, in tackling a binary classification problem. The following points summarize the key findings and insights:

4.1 Summary of Findings

- **Training Process:** The training loss curve showed a rapid decline in the initial epochs, followed by a gradual flattening, indicating that the model successfully optimized its parameters and converged to a minimal loss.
- **Model Performance:** The model achieved an accuracy of 97.5%, as demonstrated by the confusion matrix. This high accuracy highlights the model's ability to generalize well to unseen data, with minimal false predictions.
- **Prediction Example:** A specific prediction example showcased the model's ability to classify inputs with high confidence (probability of 0.9996), further validating its robustness.

4.2 Insights

- The simplicity of the model architecture, consisting of a single hidden layer with 32 neurons, was sufficient to achieve high accuracy for this binary classification task. This suggests that for similar tasks, overly complex architectures may not be necessary.
- The importance of preprocessing steps, such as normalization and class balancing, was evident in ensuring stable and effective training.

- The project underscored the value of building neural networks from scratch, providing deeper insights into the underlying mechanics of forward and backward propagation, gradient descent, and parameter updates.

4.3 Future Work

While the results are promising, there are opportunities for improvement and further exploration:

- **Testing on a Larger Dataset:** Expanding the dataset could help evaluate the model's scalability and robustness.
- **Incorporating Additional Metrics:** Metrics such as precision, recall, and F1-score could provide a more comprehensive understanding of the model's performance.
- **Exploring Alternative Architectures:** Experimenting with deeper networks or different activation functions may further improve performance.
- **Optimization Techniques:** Investigating advanced optimization methods, such as Adam or RMSProp, could accelerate convergence and enhance results.

4.4 Conclusion

This project successfully implemented a neural network from scratch, achieving high accuracy and demonstrating the power of deep learning for binary classification tasks. By understanding and implementing the core principles of neural networks, this work lays the foundation for exploring more complex architectures and applications in future projects. .

5 Code

The code for this project is available in a Google Colab notebook. You can access it using the following link:

Google Colab Notebook - Project Code

References

[1] The NumPy Community, *NumPy Documentation*, 2024. `https://numpy.org/doc/`

[2] The Matplotlib Development Team, *Matplotlib Documentation*, 2024. `https://matplotlib.org/stable/contents.html`

[3] Google, *Google Colaboratory*, 2024. `https://colab.research.google.com`