Report on Multi-Layer Neural Network Implementation for Classification

IN3063: Mathematics and Programming for AI Coursework

GROUP 15- Aziz Kilic, Alice Khin, Oliver Buterbaugh, Jokubas Stankaitis, Ibhaan Kudalkar

GitHub Link:

https://github.com/arachellee/IN3063-Group-15/blob/main/Prog%20and%20Maths%20for%20AI.ipynb

Introduction

This report describes the creation and evaluation of a multi-layered neural network for classification tasks using the CIFAR-10 dataset. The project involved coding components from scratch using libraries like NumPy and matplotlib, torch, torchvision and pandas. Key tasks included dataset exploration, activation functions, dropout regularization, optimization methods, and hyperparameter tuning. The project aimed to understand neural network design and performance, using the CIFAR-10 dataset for team-based classification problems.

# Dataset Selection and Description

The dataset chosen was CIFAR-10, as it was relatively easy and was suitable for classification models evaluation. The dataset contains 60,000 32x32 colour images, for 10 classes: 50,000 training and 10,000 test images. All classes are objects (in the real world) (eg aeroplanes, cars, birds, cats, etc).

Our decision to use CIFAR-10 was based on the low image size which will have reasonable computational footprint but pose a relatively difficult classification task. The diversity in the dataset also ensures robustness in evaluating the implemented neural network.

# Implementation details

**Code and Methodology**

**Data Preparation**

1. **Dataset**: CIFAR-10 dataset, consisting of 60,000 32x32 color images in 10 classes.
2. **Preprocessing**:
   * Data was flattened into vectors.
   * Pixel values were normalized to the range [0, 1].
   * The dataset was split into training (80%), validation (10%), and testing (10%) sets.
3. **One-hot Encoding**: Labels were converted into one-hot vectors for use in the softmax classifier.

**Neural Network Architecture**

The network was implemented with the following components:

* **Hidden Layers**: Configurable hidden layers with ReLU, Sigmoid, or dropout applied.
* **Output Layer**: Softmax activation for multi-class classification.
* **Optimizers**: Variants of SGD with and without momentum.

**Variants Tested**

1. **Default Configuration**: Three layers with ReLU activation and softmax output.
2. **Two Sigmoid Layers and One ReLU**: Explores performance with mixed activations.
3. **Three ReLU Layers with SGD Optimizer**: Default optimizer without momentum.
4. **Three Sigmoid Layers with SGD and Momentum**: Added momentum to enhance optimization.
5. **Three ReLU Layers with Dropout**: Introduced dropout to test overfitting control.
6. **Single Sigmoid Layer**: Tested network performance with reduced depth.

**Results and Observations**

**Default Configuration (3 ReLU Layers)**

* **Loss**: Gradual reduction over epochs.
* **Accuracy**: Achieved stable training and validation accuracy.
* **Graph Analysis**:
  + Loss curve demonstrated smooth convergence.
  + Accuracy curve reflected consistent learning.

**Two Sigmoid, One ReLU**

* **Observation**: Slower convergence compared to ReLU-only setups.
* **Reason**: Sigmoid’s vanishing gradient effect hindered optimization.
* **Output**: Achieved reasonable accuracy, though inferior to ReLU-dominant models.

**Three ReLU Layers with SGD**

* **Observation**: Baseline performance.
* **Output**: Delivered reliable results without significant tuning.

**Three Sigmoid Layers with Momentum**

* **Observation**: Faster convergence than standard SGD due to momentum.
* **Output**: Improved accuracy compared to two-sigmoid setup.

**Three ReLU Layers with Dropout**

* **Observation**: Dropout helped mitigate overfitting, particularly noticeable in validation loss.
* **Output**: Achieved balanced generalization.

**Single Sigmoid Layer**

* **Observation**: Performance drop due to reduced network capacity.
* **Output**: Highlighted the importance of network depth for complex datasets like CIFAR-10.

**Fixed Seed for Reproducibility**

Throughout the experiments, a fixed seed ensured consistent weight initialization, random shuffling, and validation splits. This approach eliminated variability between runs, enabling fair comparisons across configurations.

**Conclusion**

The experiments demonstrated the impact of architectural and optimization choices on neural network performance. Key takeaways include:

1. **Activation Functions**: ReLU provided robust performance, while Sigmoid struggled due to vanishing gradients.
2. **Optimizers**: Momentum accelerated convergence without requiring higher learning rates.
3. **Dropout**: Effectively reduced overfitting, especially for deeper networks.
4. **Depth**: Adequate hidden layers are essential for handling complex data.

**References**

Appendix 1:

**A black screen with white text

Description automatically generated**

Appendix 2:

A collage of images of animals

Description automatically generated

Appendix 3:

A screen shot of a black screen

Description automatically generated

Appendix 4:

A graph with a line

Description automatically generated

Appendix 5:

A graph showing a line

Description automatically generated

Appendix 6:

A black screen with white text

Description automatically generated

Appendix 7:

A graph with a line

Description automatically generated

Appendix 8:

A screen shot of a black screen

Description automatically generated

Appendix 9:

A screenshot of a computer screen

Description automatically generated

Appendix 10:

A graph with a line and a red line

Description automatically generated