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 details the implementation and evaluation of a multi-layered neural network designed for classification tasks using the CIFAR-10 dataset. The project required implementing various components of a neural network from scratch using many libraries like NumPy, matplotlib etc for the neural network’s core computations. Key sub-tasks included dataset exploration, implementing activation functions such as Sigmoid, ReLU, and SoftMax, dropout regularisation, a fully parametrizable neural network class, optimization methods, and hyperparameter tuning.

The aim was to understand the intricacies of designing a neural network while exploring different configurations and observing their effects on classification performance. By using CIFAR-10, a relatively complex image dataset, the project aligns with the goal of solving challenging classification problems within a team environment.

# Dataset Selection and Description

The CIFAR-10 dataset was chosen due to its relative complexity and its suitability for evaluating classification models. The dataset consists of 60,000 32x32 colour images across 10 classes, with 50,000 training images and 10,000 test images. Each class represents a real-world object (e.g. aeroplanes, cars, birds, cats, etc).

The choice of CIFAR-10 was motivated by its small image size, which allows for manageable computational requirements while still presenting a non-trivial classification challenge. The diversity in the dataset also ensures robustness in evaluating the implemented neural network.

# Implementation details

**a. Activation Functions: Sigmoid and ReLU**

Activation functions are critical in introducing non-linearity to neural networks, enabling them to learn and model complex patterns in the data. In this implementation, the Sigmoid and ReLU (Rectified Linear Unit) activation functions were chosen due to their distinct properties and advantages.

**Sigmoid Activation Function**

The Sigmoid function was implemented for both the forward and backward passes. Mathematically, the sigmoid function is defined as:

This function maps any real-valued number to the range (0, 1), making it particularly suitable for probabilistic interpretations in machine learning tasks. In the forward pass, this transformation ensures that the output values are bounded, avoiding extreme values that could destabilize the training process.

For the backward pass, the derivative of the sigmoid function was derived as:

The implementation of this derivative is efficient and supports smooth backpropagation through the network.

**Advantages of Sigmoid:**

1. **Probabilistic Interpretation:** The sigmoid function is widely used in binary classification tasks and as an activation function in output layers for probability estimation.
2. **Smooth Gradient:** Its continuous and smooth curve ensures stable gradient flow during backpropagation for smaller networks.

**Disadvantages of Sigmoid:**

1. **Vanishing Gradients:** For large positive or negative input values, the gradient approaches zero, which can significantly slow down learning in deeper networks.
2. **Output Saturation:** When inputs lie in the saturation regions (close to 0 or 1), small changes in input result in negligible changes in output, hindering effective learning.

**ReLU Activation Function**

The ReLU (Rectified Linear Unit) activation function was implemented for both forward and backward propagation. ReLU is defined as:

During the forward pass, ReLU outputs the input directly if it is positive; otherwise, it outputs zero. This simplicity makes ReLU computationally efficient compared to sigmoid or tanh functions. For the backward pass, the derivative of ReLU is:

This derivative ensures efficient computation of gradients, allowing deeper networks to converge faster.

**Advantages of ReLU:**

1. **Efficient Computation:** ReLU's simplicity significantly reduces computational overhead, making it well-suited for large networks.
2. **Non-Linearity:** By introducing non-linearity, ReLU enhances the learning capacity of the network.
3. **No Vanishing Gradient:** Unlike sigmoid, ReLU does not saturate for positive inputs, ensuring gradient flow is maintained in deeper layers.

**Disadvantages of ReLU:**

1. **Dying ReLU Problem:** Neurons can sometimes output zero for all inputs, effectively rendering them inactive and reducing network capacity.
2. **Unbounded Output:** ReLU's unbounded nature can lead to exploding gradients in certain scenarios.

In this project, ReLU was primarily used in the hidden layers due to its superior performance in mitigating the vanishing gradient problem and ensuring efficient learning in deep networks.

**b. Softmax Layer**

The softmax function was implemented as the final activation layer of the neural network, converting raw logits into probabilities for multi-class classification. The softmax function is defined as:

where represents the logits for a specific class and is the total number of classes.

In the forward pass, the implementation used a numerical stability technique to prevent overflow issues during the exponential computation. Specifically, the maximum logit value was subtracted from all logits before applying the exponential function:

This adjustment ensures that the exponential terms remain within a manageable range, avoiding numerical instability.

For the backward pass, the derivative of the softmax function was computed using its analytical gradient. This derivative ensures accurate gradient flow for updating the weights and biases during backpropagation.

**Advantages of Softmax:**

1. **Probabilistic Output:** Softmax outputs a probability distribution, making it ideal for multi-class classification tasks.
2. **Differentiability:** Its continuous and differentiable nature supports effective backpropagation.

**Numerical Challenges:**

Without the stability trick, the exponential computation could result in overflow errors, particularly when logits have large values. The implemented solution ensures robust computation even in such scenarios.

**c. Dropout Regularization**

Dropout was implemented to address overfitting and enhance the network's generalization capabilities. The inverted dropout technique was used, where neuron activations were scaled during training to maintain consistent outputs during inference.

**Forward Pass**

In the forward pass, a binary mask was generated, with each element randomly set to 0 or 1 based on the specified dropout rate . The activations were then scaled by to ensure that the expected sum of activations remained unchanged:

**Backward Pass**

During the backward pass, the same dropout mask was applied to the gradients, ensuring that only the active neurons contributed to the gradient computation.

**Advantages of Dropout:**

1. **Prevents Overfitting:** By deactivating neurons randomly, dropout prevents the network from relying too heavily on specific neurons, improving generalization.
2. **Encourages Robustness:** Dropout forces the network to distribute learning across neurons, enhancing robustness.

**Inverted Dropout:**

The inverted dropout technique ensures that the forward pass remains unchanged during inference, simplifying implementation and avoiding discrepancies between training and testing phases.

**d. Fully Parametrizable Neural Network Class**

A fully connected neural network class was implemented, allowing the following parameters to be adjusted:

* **Number of hidden layers:** Users can specify the depth of the network.
* **Units per layer:** Fully flexible to define the number of neurons in each layer.
* **Activation functions:** Sigmoid, ReLU, or others can be applied layer-wise.
* **Dropout and Regularization:** Supports inverted dropout and L2 regularization.

**Parameters Used**

* **Learning Rate:** 0.01 (tuned via grid search).
* **Batch Size:** 64 (mini-batch gradient descent).
* **Epochs:** 20.
* **Regularization:** L2 regularization with a penalty factor of 0.001.

**e. Optimization Methods**

**Stochastic Gradient Descent (SGD)**

The basic SGD optimizer was implemented to update weights and biases. The simplicity of this method makes it a foundational optimization technique.

**SGD with Momentum**

Momentum was introduced to accelerate convergence by incorporating a velocity term that accumulates gradients over iterations. The velocity term helps escape shallow minima and improves training stability.

**Training and Validation**

The network was trained on the CIFAR-10 dataset using the hyperparameters outlined above. The training loss and accuracy were monitored across epochs, and the following observations were made:

1. **Activation Function Impact:**
   * Networks with ReLU activation demonstrated faster convergence compared to Sigmoid.
   * Sigmoid networks suffered from slower training due to vanishing gradients in deeper layers.
2. **Dropout Effectiveness:**
   * Dropout improved test performance by reducing overfitting. Higher dropout rates, however, led to underfitting.
3. **Optimizer Comparison:**
   * SGD with Momentum outperformed basic SGD, achieving lower loss and higher accuracy within fewer epochs.