Lab: Implementing Gradient Descent for a Single-Layer Perceptron

1 Introduction

In this lab, you will implement **Gradient Descent** to train a **single-layer perceptron** for binary classification on simulated data. The perceptron is one of the simplest neural network models, and understanding its mechanics will provide a foundation for more complex machine learning models.

2 Background

2.1 What is Gradient Descent?

Gradient descent is an optimization algorithm used to minimize functions by iteratively moving in the direction of the steepest descent of the function. It is widely used in machine learning to optimize model parameters by minimizing a defined error or **loss function**.

For a given parameter θ, the gradient descent update rule is:

$$\theta := \theta - \alpha \frac{\partial L}{\partial \theta}$$

where:

- α is the **learning rate**, which controls the step size.
- L is the **loss function**, measuring the error between predictions and actual values.
- $\frac{\partial L}{\partial \theta}$ is the partial derivative of the loss function with respect to θ.

By iteratively applying this update rule, gradient descent minimizes the loss function, moving towards optimal parameter values.

2.2 What is a Single-Layer Perceptron?

A single-layer perceptron is a simple linear classifier that maps input features to a binary output. It consists of a weight vector $\mathbf{w}$ and a bias b, and computes the output as follows:

$$\hat{y} = f(\mathbf{w} \cdot \mathbf{x} + b)$$

where:

- $\mathbf{w} \cdot \mathbf{x}$ is the **dot product** of the weight vector and the input vector.
- b is the **bias** term, allowing the decision boundary to shift.
- $f(z)$ is an **activation function** applied to the result.

In this lab, we will use the **sigmoid activation function**, which outputs probabilities between 0 and 1:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

For binary classification, our perceptron will output a label $\hat{y}$ of 1 if $\sigma(z) \geq 0.5$ and 0 otherwise. We train the perceptron by finding the weights and bias that minimize the **binary cross-entropy loss function**, which measures the difference between predicted probabilities and actual labels.

2.3 Loss Function

The binary cross-entropy loss function for a single-layer perceptron is:

$$L = -\frac{1}{m} \sum_{i=1}^{m} \left(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right)$$

where:

- m is the number of samples.
- y_i is the true label for sample i (0 or 1).
- $\hat{y}_i$ is the predicted probability for sample i, given by the sigmoid function $\sigma(\mathbf{w} \cdot \mathbf{x}_i + b)$.

3 Gradient Descent for Training the Perceptron

To minimize the binary cross-entropy loss, we use gradient descent to update the weights $\mathbf{w}$ and bias b.

- Compute the gradients for $\mathbf{w}$ and b:

$$\frac{\partial L}{\partial w_j} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)x_{i,j}$$

$$\frac{\partial L}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)$$

  where:

  - $x_{i,j}$ is the j-th feature of the i-th sample.
  - $\hat{y}_i = \sigma(\mathbf{w} \cdot \mathbf{x}_i + b)$ is the predicted probability.

- Update the weights and bias with the gradients:

$$w_j := w_j - \alpha\frac{\partial L}{\partial w_j}$$

$$b := b - \alpha\frac{\partial L}{\partial b}$$

Summary of Gradient Descent Steps 1. Calculate $\hat{y}_i$ for each sample using the current $\mathbf{w}$ and b. 2. Compute the gradients of L with respect to $\mathbf{w}$ and b. 3. Update $\mathbf{w}$ and b using the learning rate α. 4. Repeat until the loss stabilizes or a set number of iterations is reached.

4 Tasks

Complete the following tasks to implement a single-layer perceptron with gradient descent.

1. **Generate and Split the Dataset.**

- Generate a synthetic dataset with 1000 samples and two features. Ensure the features have some separation between the two classes, for example by generating data with two clusters.
- Label each sample with 0 or 1 based on its class.
- Split the data into 80% training and 20% testing sets.

2. **Initialize Parameters.**

   - Set the weights $\mathbf{w}$ and bias b to small random values (e.g., initialized to zeros or small random numbers).

3. **Define the Activation Function.**

   - Implement the sigmoid activation function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

   - Use a threshold of 0.5 on $\sigma(z)$ to classify outputs as 0 or 1.

4. **Implement Gradient Descent.**

   - Calculate the loss using binary cross-entropy on the training set.
   - Compute the gradients for $\mathbf{w}$ and b as shown above.
   - Update the weights and bias based on the gradients and the learning rate α.
   - Repeat for a fixed number of iterations (e.g., 1000) or until the loss stabilizes.

5. **Evaluate the Model.**

   - After training, evaluate the final loss on both the training and testing sets.
   - Use the model to predict labels on the testing set, and calculate the classification accuracy.

5 General Instructions

- Experiment with different learning rates to observe their impact on convergence.
- Track and plot the loss over iterations to visualize the optimization process.
- Use a random seed for reproducibility when generating the dataset.

6 Expected Results

At the end of the lab, you should have:

- A trained single-layer perceptron using gradient descent.
- Plots showing the decrease in loss over iterations.
- The training and testing loss values, along with the testing accuracy.
- Observations on the effect of the learning rate and number of iterations on the model's performance.