|  |  |
| --- | --- |
| **Ex No: 1**  **Date: 13/08/24** | **Gradient descent implementation** |

**Objective:**

To build a logistic regression classifier to recognize Cat’s vs Non cat images using Gradient descent implementation.

**Descriptions:**

Binary classification is classifying elements of a given set into two groups. Logistic regression is an algorithm for binary classification. We have an input image x and the output y is a label to recognize the image. 1 means cat is in an image, 0 means that a non-cat object is in an image**.**



Logistic regression is a supervised learning algorithm that we can use when labels are either 0 or 1 and this is the so-called Binary Classification Problem. An input feature vector X may correspond to an image we want to recognize as a cat picture (1) or a non-cat picture (0). That is, we want an algorithm to output the prediction which is an estimate of y: Logistic Regression doesn't have a hidden layer. If you initialize the weights to zeros, the first example x fed in the logistic regression will output zero but the derivatives of the Logistic Regression depend on the input x (because there's no hidden layer) which is not zero. So at the second iteration, the weights values follow x's distribution and are different from each other if x is not a constant vector.

**Model:**

https://piehqfpkotypaenhpraenx.coursera-apps.org/notebooks/Week%202/Logistic%20Regression%20as%20a%20Neural%20Network/images/LogReg_kiank.pnghttps://piehqfpkotypaenhpraenx.coursera-apps.org/notebooks/Week%202/Logistic%20Regression%20as%20a%20Neural%20Network/images/LogReg_kiank.png****

**Building the parts of the algorithm**

The main steps for building a Neural Network are:

1. Define the model structure (such as the number of input features)  
   The number of features is given by the shape of the training data.

**#code**

**m\_train=train\_set\_x\_orig.shape[0]**

1. Initialize the model's parameters

Model parameters for logistic regression are set randomly and hence the weights are usually initialized to zero or smaller values and the bias to 0.

**#code**

**w=np.zeros((dim,1))**

**b=0**

1. Loop:
   * Calculate current loss (forward propagation)

Forward propagation involves calculating the model's output using the current parameters. The output is compared with the true labels to compute the loss

cf = 1 / (1 + np.exp(-(np.dot(w.T, X) + b)))

cost = -1/m \* np.sum(Y \* np.log(cf) + (1 - Y) \* np.log(1 - cf))

* + Calculate current gradient (backward propagation)

Backward propagation involves calculating the gradient of the loss with respect to the model parameters. These gradients indicate the direction in which each parameter should be adjusted to minimize the loss.

dw = 1/m \* np.dot(X, (cf - Y).T)

db = 1/m \* np.sum(cf - Y)

* + Update parameters (gradient descent)  
    Update the model parameters using the gradients computed during backward propagation. This is done using the gradient descent algorithm, which adjusts the parameters in the direction that reduces the loss.

w = w - learning\_rate \* dw

b = b - learning\_rate \* db

**GitHub Link:**

* **https://github.com/LohithR22/Fundamentals\_of\_DeepLearning**