NEURAL NETWORKS AND DEEP LEARNING CST 395 CS 5TH SEMESTER HONORS COURSE- Dr Binu V P, 9847390760

CS 5th Semester Honors course for the Computer Science at KTU- Dr Binu V P

Regression with Gradient descent-Python code

September 08, 2022

Optimizing parameters is the ultimate goal of every machine learning algorithm. You want to get the optimum value of the slope and the intercept to get the line of best fit in linear regression problems. You also want to get the optimum value for the parameters of a sigmoidal curve in logistic regression problems. Gradient Descent does it all

Loss Function: A function that returns the cost associated with the model and measures how well our model is doing on the training data. If the cost is too high, it means that the predictions by our model are deviating too much from the observed data. In any machine learning algorithm, our ultimate mission is to minimize the loss function. Various loss functions which we use are:

Regression Losses:
L1 Loss / Mean Absolute Error
L2 Loss / Mean Squared Error
Root Mean Squared Error

Classification Losses:
Log Loss (Cross-Entropy Loss)
SVM Loss (Hinge Loss)

Learning Rate: This is the hyperparameter that determines the steps the gradient descent algorithm takes. Gradient Descent is too sensitive to the learning rate. If it is too big, the algorithm may bypass the local minimum and overshoot. If it too small, it might increase the total computation time to a very large extent. We will see the effect of the learning rate in depth later in the article.

Gradient: Basically, it is a measure of the steepness of a slope. And technically, when we sum up all the first-order derivatives of all the variables in a function, it gives us gradient. For example, if we consider linear regression, we have two parameters, slope, and the intercept, to minimize. So, we calculate derivatives w.r.t. both slope & the intercept and then sum them up to get the gradient for it.

Descent: To optimize parameters, we need to minimize errors. The aim of the gradient descent algorithm is to reach the local minimum (though we always aim to reach the global minimum of the function. But if a gradient descent algorithm once attains the local minimum, it is nearly impossible to reach the global minimum.). The algorithm accomplishes this by an iterative process of calculating step size at every iteration. And, this iterative calculation of step size to reach a local minimum (or in other words, descending to the point of minimum) is known as the descent (Enough of that going down the hill example).

Lets consider a linear model. The hypothesis function can be written as
$h(\theta) = \theta_0 + \theta_1 X$

We has seen how to find the parameters θ_0 and θ_1 analytically.It is because the analytical method turns out to be hugely computationally expensive when we have a dataset with millions of data points. Gradient descent, on the other hand, gives us similar results while minimizing the computation time massively.

Mathematically, the cost function and the gradient can be represented as follows:
Cost Function: $J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h(\theta)^i - y^i)^2$
Gradient : $\delta J(\theta)/\delta\theta = \sum_{i=1}^{m} (h(\theta)^i - y^i) X_j^i$

Intuition Behind the Cost Function

Our goal here is to minimize the cost function in a way that it comes as close to zero as possible. Having a high negative value is also as bad as a high positive value for the cost function. So, in order to keep the value of cost function >=0, we are squaring it up. We could have done the same using absolute but there are two main reasons why we are not doing so.

1.Putting absolutes instead of squaring it would penalize the model equally for high and low residuals, whereas we need a weighted penalizing rule where the data points with higher residuals are penalized more and lower ones less. That can be simply achieved by squaring the residuals/errors.

2.Also, according to the Gaussian Noise concept, there are two kinds of errors, systematic and random. Simply stated, a systematic error is something that follows a certain direction. It is consistent in nature, and predictable. On the other hand, a random error is noise in our distribution. Once we've taken care of the systematic noise component, the best predictor is obtained when the random noise is minimized. Putting it another way, the best predictor is the one with the tightest distribution (smallest variance) around the predicted value. Minimizing the least squared loss is the same thing as minimizing the variance! That explains why the least squared loss works for a wide range of problems. The underlying noise is very often Gaussian, because of the CLT (Central Limit Theorem), and minimizing the squared error turns out to be the right thing to do!

We are putting in ½ so that the '2' coming from the derivation of the cost function while minimizing it for calculating gradient, cancels out, and makes gradient more concise. Anyway, it is a constant does not matter in calculating theta's value (as we are differentiating it).

Also, we are averaging the function over m, which is the total number of data points in the training dataset. This is so that the value of the calculated gradient does not change the scale and always averages out to a central value in case a new data point comes in.

Implementation With Gradient Descent

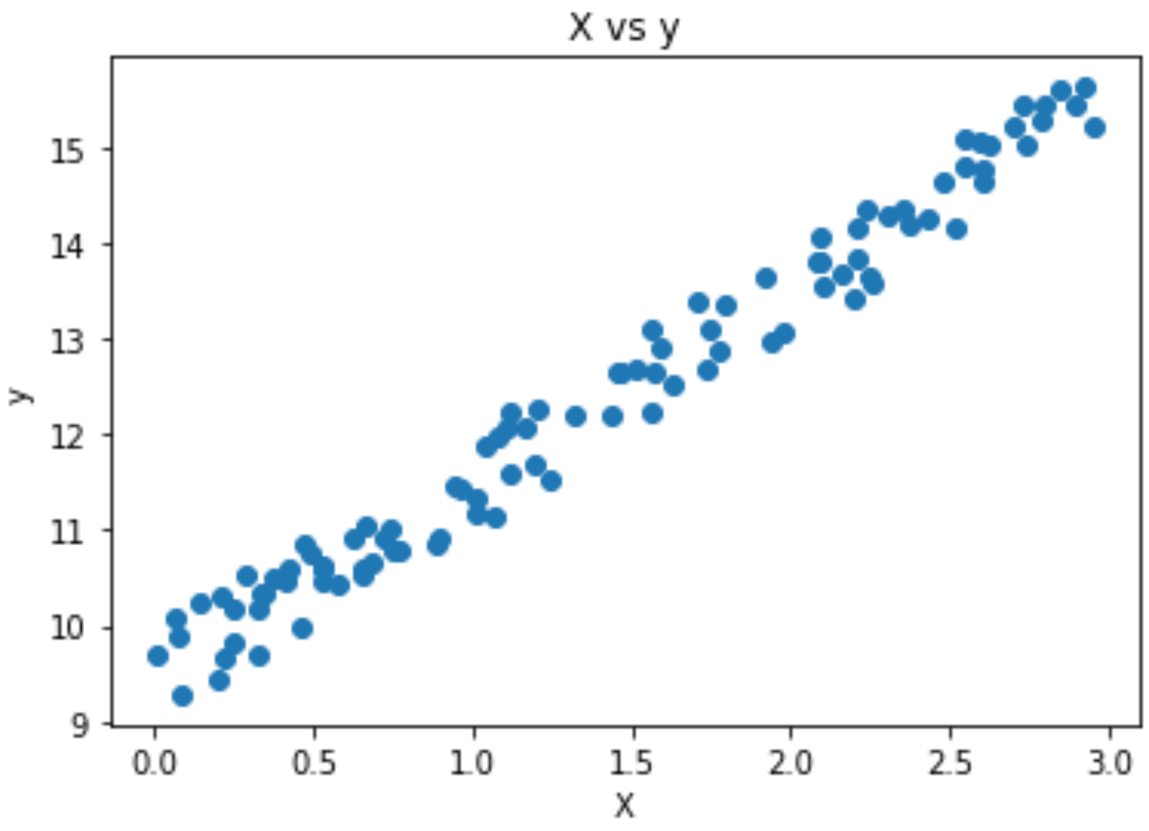
Before we implement gradient descent, knowing the intuition behind it is a must. Since we are now done with that part, let's dive into the actual implementation of it.

We want to calculate the value for θ_0 and θ_1 but we can have multiple features (>=2). In that case, the general formula to calculate consecutive step sizes will be
$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h(\theta)^i - y^i) X_j^i$

where α is the learning rate. We can now infer from the formula that the α influences the size of the step a lot as we are multiplying the gradient with the alpha for every iteration.

Enough of mathematics, let us start with the actual code. Let's generate a randomized dataset first using the NumPy's random function and plot it to visualize our dataset distribution with a scatter plot.

```
# Importing Libraries
import numpy as np
import matplotlib.pyplot as plt
# Generating Randomized dataset
X = 3*np.random.rand(100,1)
y = 9 + 2*X+np.random.rand(100,1)
# Scatter plot
plt.scatter(X,y)
plt.xlabel('X')
plt.ylabel('y')
plt.title('X vs y')
plt.figure(figsize=(15,25))
```



Now we will initialize random values for the parameters to be optimized, θ_0 and θ_1. We will write two functions to calculate cost and gradient descent by iterating and store them in two distinct NumPy arrays.The above-mentioned formulae have been used in calculating the cost and the consecutive values for θ using the gradient.

```
def cost(theta,X,y):
    ''' Calculates cost of the function. X & y have their usual meaning. theta - vector of coefficients. '''
    m = len(y)
    # Calculating Cost
     c = (1/2*m) * np.sum(np.square((X.dot(theta))-y))
    return c
def gradient_descent(X,y,theta,alpha,iterations):
    ''' returns array of thetas, cost of every iteration
        X - X matrix with added bias.
        y - target variable matrix
        theta - matrix of regression coefficients
        alpha - learning rate
        iteration - number of iteration to be run '''
#Getting number of observations.
        m = len(y)
# Initializing cost and theta's arrays with zeroes.
        thetas = np.zeros((iterations,2))
        costs = np.zeros(iterations)
# Calculating theta for every iteration.
        for i in range(iterations):
          theta = theta - (1/m)*alpha*(X.T.dot((X.dot(theta))-y))
        thetas[i,:] = theta.T
        costs[i] = cost(theta,X,y)
        return theta,thetas,costs
# Learning Rate
alpha = 0.01
# Number of iterations
iterations = 3000
# Initializing a random value to give algorithm a base value.
theta = np.random.randn(2,1)
# Adding a biasing constant of value 1 to the features array.
X_bias = np.c_[np.ones((len(X),1)),X]
# Running Gradient Descent
theta,thetas,costs = gradient_descent(X_bias,y,theta,alpha,iterations)
# printing final values.
print('Final Theta 0 value: {:0.3f}\nFinal Theta 1 value: {:0.3f}'.format(theta[0]
[0],theta[1][0])) print('Final Cost/MSE(L2 Loss) Value: {:0.3f}'.format(costs[-1]))
```

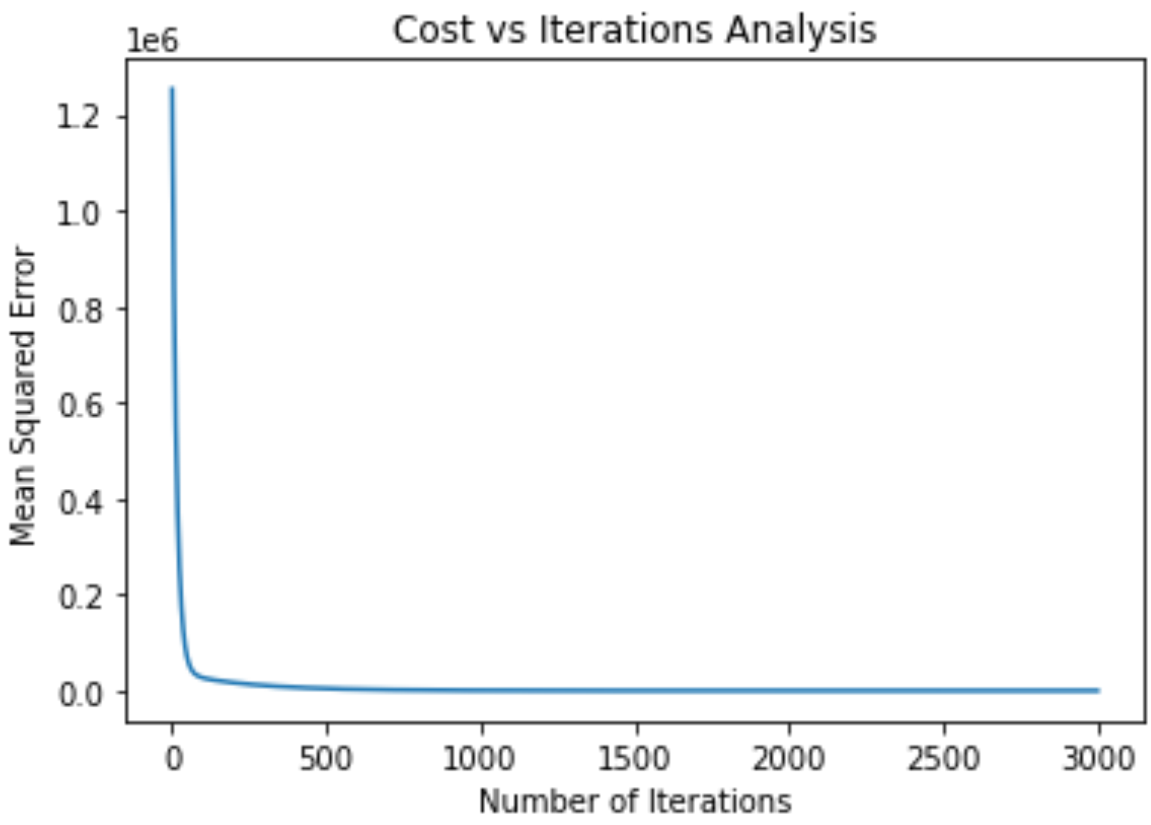
Output

```
Final Theta 0 value: 9.477
Final Theta 1 value: 2.041
Final Cost/MSE(L2 Loss) Value: 403.634
```

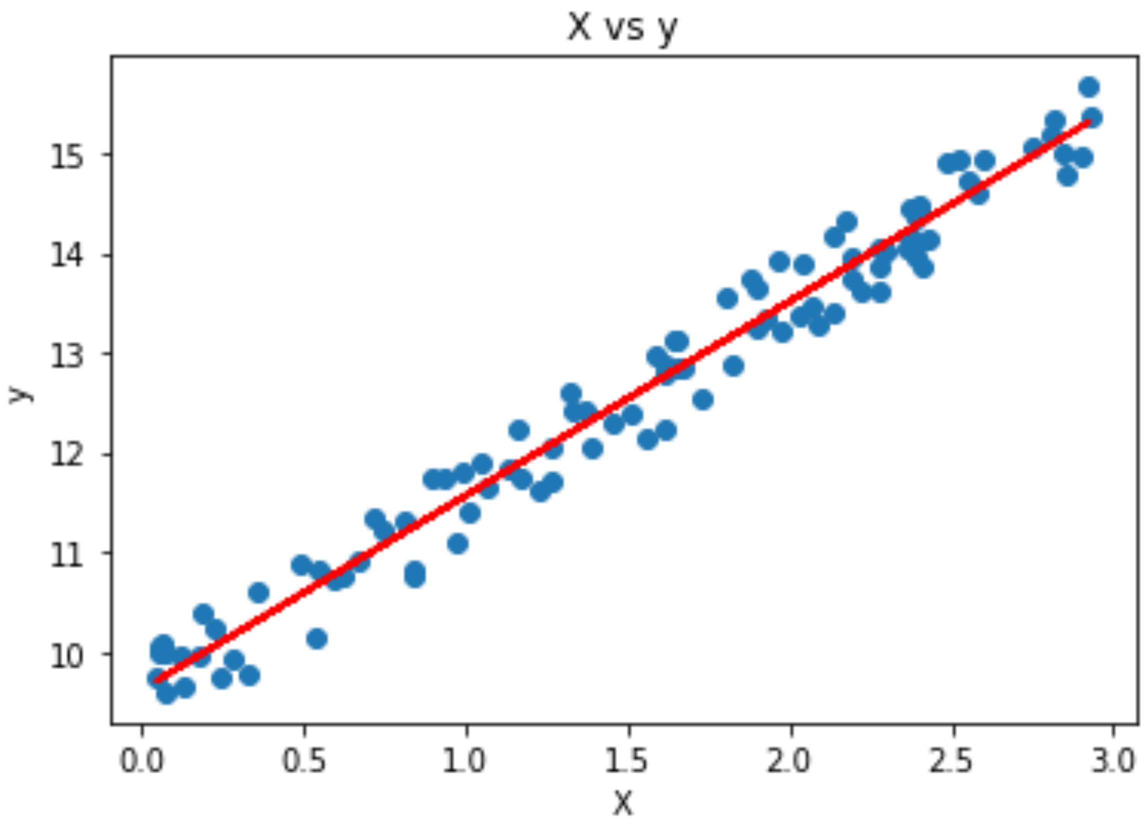
Analysis

Let us check how the L2 Loss reduces along with increasing iterations by plotting a graph.

```
# Plotting Line Plot for Number of Iterations vs MSE
plt.plot(range(iterations),costs)
plt.xlabel('Number of Iterations')
plt.ylabel('Mean Squared Error')
plt.title('Cost vs Iterations Analysis')
```



The best fit regression line obtained using the gradient Descent method is shown in red color



Small code/Data set for testing

```
def predict(row, coefficients):
    yhat = coefficients[0]
    for i in range(len(row)-1):
        yhat += coefficients[i + 1] * row[i]
    return yhat

# Estimate linear regression coefficients using stochastic gradient descent
def coefficients_sgd(train, l_rate, n_epoch):
    coef = [0.0 for i in range(len(train[0]))]
    for epoch in range(n_epoch):
        sum_error = 0
        for row in train:
            yhat = predict(row, coef)
            error = yhat - row[-1]
            sum_error += error**2
            coef[0] = coef[0] - l_rate * error
            for i in range(len(row)-1):
                coef[i + 1] = coef[i + 1] - l_rate * error * row[i]
        print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l_rate, sum_error))
    return coef

# Calculate coefficients
dataset = [[1, 1], [2, 3], [4, 3], [3, 2], [5, 5]]
l_rate = 0.001
n_epoch = 150
coef = coefficients_sgd(dataset, l_rate, n_epoch)
print(coef)
```

Popular posts from this blog

NEURAL NETWORKS AND DEEP LEARNING CST 395 CS 5TH

SEMESTER HONORS COURSE NOTES - Dr Binu V P, 9847390760

October 03, 2022

About Me Syllabus Question Paper Dec 2022 Module 1 (Basics of Machine Learning) Overview of Machine Learning Machine Learning Algorithm Linear Regression Capacity, Overfitting and Underfitting Regularization Hyperparameters and Validatioi …

READ MORE

Machine Learning Algorithm

October 01, 2022

A machine learning algorithm is an algorithm that is able to learn from data. But what do we mean by learning? Mitchell (1997) provides the definition “A computer program is said to learn from experience E with respect to some class of tasks T and …

READ MORE

Syllabus

October 01, 2022

Syllabus Module - 1 (Basics of Machine Learning) Machine Learning basics - Learning algorithms - Supervised, Unsupervised, Reinforcement, overfitting, Underfitting, Hyper parameters and Validation sets, Estimators -Bias and Variance. Challenges …

READ MORE

DR.BINU V P-9847390760

Associate Professor and Head Dept of Computer Scienc…

Karunagappally.Love to share the thoughts and views .I play lot of Games and basically an Athlet in my school and college days.I never keep my studies as a second option.Stood first In MTech and BTech.I did my research in Crypto. I

VISIT PROFILE

Archive

Report Abuse