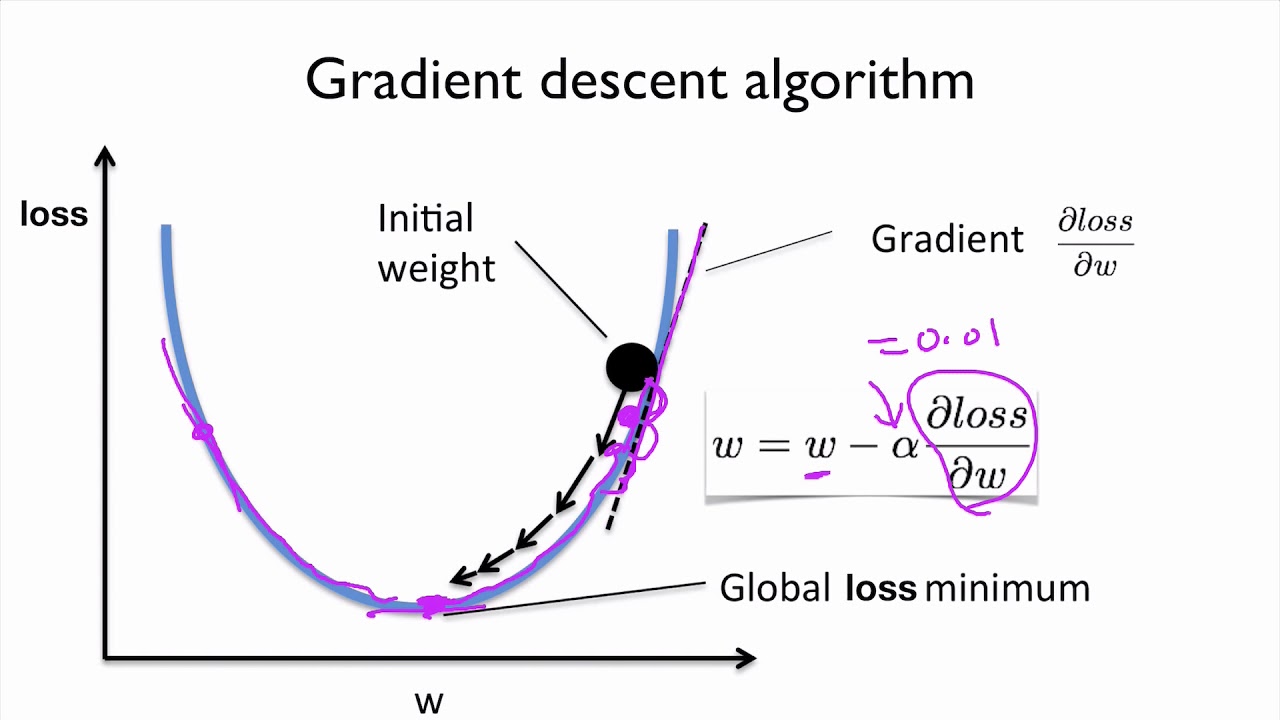
8/25/2019

**Gradient Descent Algorithm Implementation**

Linear Regression Model



Tulshi Chandra Das, BSSE 0811

## Institute of Information Technology, University of Dhaka

#### Bachelor of Science in Software Engineering

Course Code: SE-837

#### Typescript program of gradien descent

##### SUBMITTED TO

Dr. B M Mainul Hossain

Associate Professor

Institute of Information Technology

University of Dhaka.

##### SUBMITTED BY

Tulshi Chandra Das

BSSE0811

##### 

## Table of Contents

[CHAPTER-01: SOURCE CODE 1](#_Toc17592298)

[CHAPTER -02: Build And Run 4](#_Toc17592299)

[2.1 prerequisite 4](#_Toc17592300)

[2.2 Build and run 4](#_Toc17592301)

[2.3 How program works 4](#_Toc17592302)

# CHAPTER-01: SOURCE CODE

Programming Language: Typescript, Filename: gradientDescent.ts

gradientDescent.ts(1-36)

//h(x) = theta0 + theta1\*x;

interface point {

x: number,

y: number

}

class GradientDescent {

private sampleSize = 30;

private learningRate = 0.01;

private maxIteration = 1000;

private trainingData: point[] = [];

private isConverged = false;

private theta0 = 0;

private theta1 = 0;

private threshold = 0.0001;

constructor(sampleSize = 30, learningRate = 0.01) {

this.sampleSize = sampleSize;

this.learningRate = learningRate;

this.init();

}

private init() {

this.prepareTrainingData();

}

public start() {

let iteration = 1;

while (!this.isConverged && iteration < this.maxIteration) {

console.log('\niteration:' + iteration);

this.iterate();

iteration++;

}

}

gradientDescent.ts(37-69)

private iterate() {

var mse\_before = this.J();

let temp0 = this.theta0 - (this.learningRate \*

this.derivativeTheta0());

let temp1 = this.theta1 - (this.learningRate \*

this.derivativeTheta1());

this.theta0 = temp0;

this.theta1 = temp1;

console.log("theta0:" + this.theta0);

console.log("theta1:" + this.theta1);

this.isConverged = (mse\_before - this.J() < this.threshold);

}

private J() {

var sum = 0;

this.trainingData.forEach((point) => {

sum += this.squaredError(point);

});

return sum / (2 \* this.trainingData.length);

}

private squaredError(point: point) {

return Math.pow(this.predictionError(point), 2);

}

private predictionError(point: point) {

return this.h(point.x) - point.y;

}

//hypothesis

private h(x: number) {

return this.theta1 \* x + this.theta0;

}

gradientDescent.ts(70-109)

private derivativeTheta0() {

var sum = 0;

this.trainingData.forEach((point) => {

sum += this.predictionError(point);

});

return sum / this.trainingData.length;

}

private derivativeTheta1() {

var sum = 0;

this.trainingData.forEach((point) => {

sum += this.predictionError(point) \* point.x;

});

return sum / this.trainingData.length;

}

private prepareTrainingData() {

for (var i = 0; i < this.sampleSize; i++) {

var point = this.getRandomPoint(10);

this.trainingData.push(point);

}

}

private getRandomPoint(max: number): point {

let slope = 0.5;

let intercept = 2.5;

let stddev = 0.9;

let x = Math.round(Math.random() \* max);

let y = slope \* x + intercept + Math.random() \* stddev;

console.log('data points');

console.log(x + " " + y);

return {

x: x,

y: y

};

}

}

new GradientDescent().start();

.

# CHAPTER -02: Build And Run

## 2.1 prerequisite

1. Node Js
2. Typescript

The two above must be installed to build and run the program

## 2.2 Build and run

First open windows CMD or Linux terminal in the directory of gradientDescent.ts

Run the following commands:

tsc gradientDescent.ts

this will compile and create gradientDescent.js file in same directory

Now run this

node gradientDescent.js

## 2.3 How program works

Hypothesis: h(x) = theta0 + theta1\*x;

The steps are below:

1. Prepare training data set
   1. theta0 = theta0 – learningRate\*derivativeTheta0(cost function);
   2. theta1 = theta1 – learningRate\*derivativeTheta1(cost function);
2. Repeat the step 2 till converged.

Source code url:<https://github.com/TulshiDas39/gradientDescent/blob/master/gradient-descent.ts>