**Assignment 2: Gradient Descent Algorithms**

521H0489 – Hồ Hữu An

1. **The Gradient Descent algorithm for the multivariate Linear Regression problem (for multi-attribute objects).**

Collecting data will always have multiple features, not just one. Then, hθ(x) will take the form:

hθ(x1,x2,⋯,xn)=θ0 + θ1 x1 + θ2 x2 +⋯+ θn xn

With θj , xj are the weights and data of the jth feature (j runs from 0 to n). We consider x0 =1 by default because θ0 is a weight that does not depend on x, called an interceptor. Cost Function remains unchanged:

However, when upgrading the weights, we need to take the partial derivative of each θ and upgrade the weights simultaneously. That means we cannot upgrade θ1 then use the value θ1 then to calculate the partial derivative of θ2. Mathematically speaking, at each upgrade step, we need to get the gradient value at position (θ0 ,θ1 , θ2, …, θn ) then subtract the corresponding weights,

However, we can calculate the formula of in more detail. by taking partial derivatives as well. You can try taking the partial derivative yourself and compare it with the results below. We have the general formula:

, 0 ≤ j ≤ n

However, writing as below will help us visualize more clearly:

, with j = 0

, with 1 ≤ j ≤ n

Finally, getting a general Gradient Descent algorithm:

1. Start by assigning θ0 ,θ1 , θ2, …, θn random values.
2. **Simultaneous** updates of θ new by subtracting the partial derivative of each weight at its current position.

, with j = 0

, with 1 ≤ j ≤ n

1. Repeat step 2 until all calculated partial derivatives are 0 or an extremely small value.

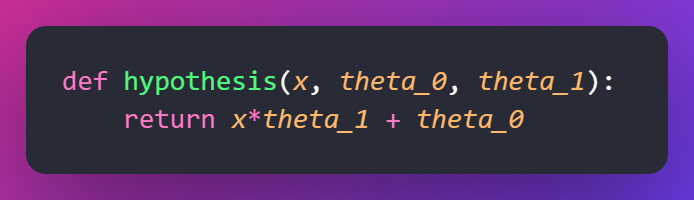
**Overview:**

* Updating the weights according to the partial derivative magnitude can cause overshooting. The best way is to control the amount of updates using the α parameter called learning rate.
* In case of using Gradient Descent with multiple weights, we update the weights simultaneously using the formula
* , with j = 0

, with 1 ≤ j ≤ n

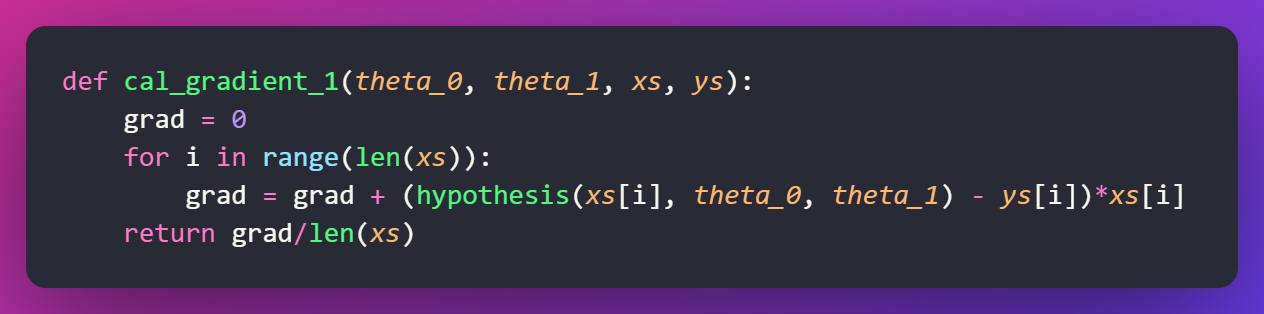
1. **Code the algorithm**

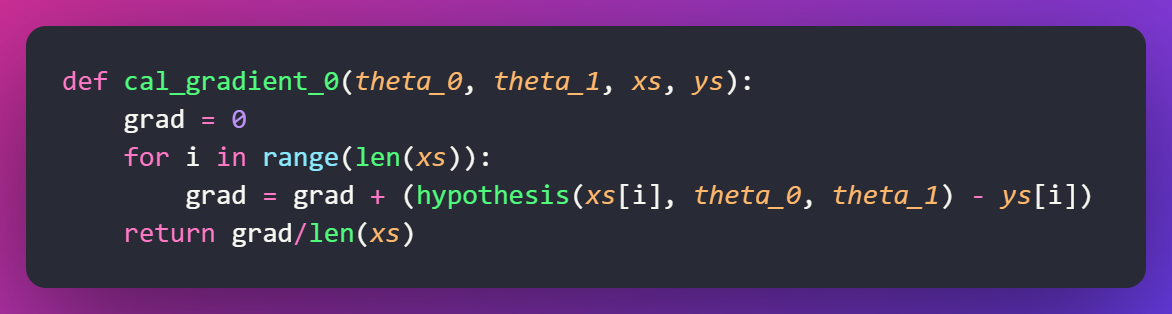
****

****

**A computer code with green and yellow text

Description automatically generated**

****

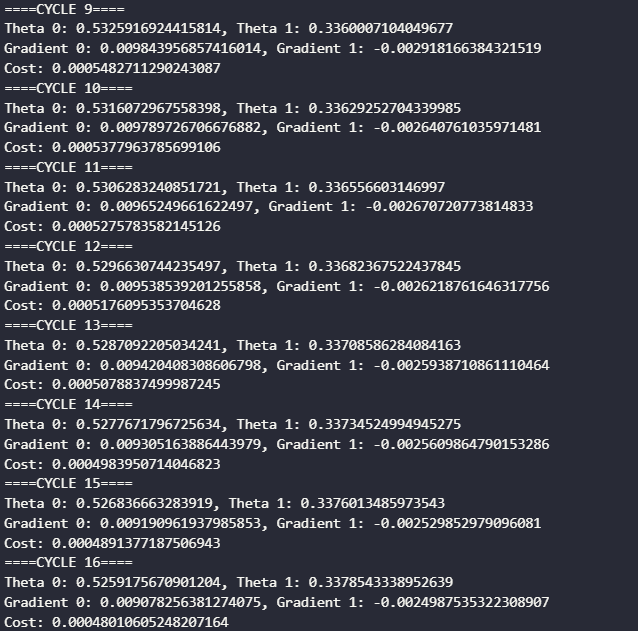
****

****

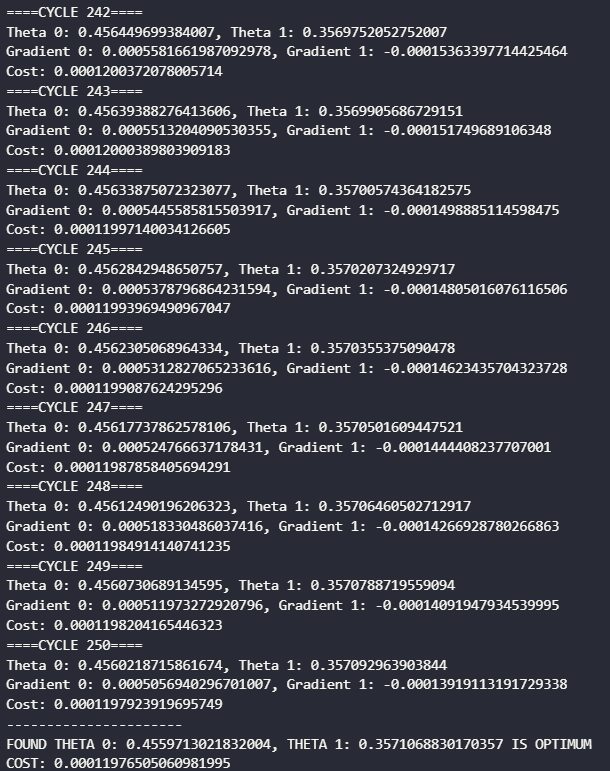
**Result:**

**A screenshot of a computer

Description automatically generated**

****

It repeats more times until optimum

****

Optimal weights

A graph of a line

Description automatically generatedThe straight line is found using Gradient Descent

1. **Different methods to determine the learning rate**
2. Trial and error method

This method is to try different values and observe their effects on the training process.

* starting with a small value, such as 0.01, and increasing or decreasing it by a factor of 10 until you find a value that works well for network
* monitoring the learning curves, which plot the loss and accuracy values against the number of epochs or iterations, to see how the network behaves with different learning rates
* checking the gradients and weights of the network to see if they are converging or exploding.

Note: The automation of this approach relies on reinforcement learning techniques like learning automata or Q-learning. By defining the continuous action space as a reinforcement learning problem and using errors to determine the reinforcement signal, it becomes possible to automate the entire process of parameter tuning.

1. Adaptive Learning Rate

This method adjusts the learning rate during training based on the characteristics of the data and the optimization process

* the initial network output and error are calculated. At each epoch new weights and biases are calculated using the current learning rate.
* New outputs and errors are then calculated

If the new error exceeds the old error by more than a predefined ratio (typically 1.04), the new weights and biases are discarded 🡺 the learning rate is decreased

Otherwise, the new weights are kept. If the new error is less than the old error, the learning rate is increased (typically by multiplying by 1.05)

1. Learning rate decay method

This method is to reduce the learning rate over time as the network gets closer to the minimum of the loss function

* starting with a relatively high learning rate to speed up the initial learning
* Then gradually lower it to avoid overshooting and oscillating around the minimum

There are different types of learning rate decay methods, such as step decay, exponential decay, inverse time decay, and adaptive decay

Note: Using a predefined schedule or a dynamic rule to adjust the learning rate according to the progress of the training.

1. Learning rate finder method

This method uses a heuristic to find a range of good learning rates for network

* starting with a very low learning rate and increase it exponentially until the loss starts to increase rapidly
* plot the loss against the learning rate and look for the point where the loss decreases the most.

This point is usually a good estimate of the optimal learning rate, or at least a lower bound for it