Why use gradient descent for linear regression, when a closed-form math solution is available?

I am taking the Machine Learning courses online and learnt about Gradient Descent for calculating the optimal values in the hypothesis.

```
h(x) = B0 + B1X
```

why we need to use Gradient Descent if we can easily find the values with the below formula? This looks straight forward and easy too. but GD needs multiple iterations to get the value.

```
B1 = Correlation * (Std. Dev. of y/ Std. Dev. of x)

B0 = Mean(Y) – B1 * Mean(X)
```

NOTE: Taken as in https://www.dezyre.com/data-science-in-r-programming-tutorial/linear-regression-tutorial

I did checked on the below questions and for me it was not clear to understand.

Why is gradient descent required?

Why is optimisation solved with gradient descent rather than with an analytical solution?

The above answers compares GD vs. using derivatives.

regression machine-learning gradient-descent

edited May 12 '17 at 4:54
smci
860 1 10 18

asked May 10 '17 at 16:52
Purus
358 1 4 6

5 You don't need gradient descent to estimate linear regression coefficients. – Sycorax May 10 '17 at 17:14

7 @Sycorax "don't need" is a strong statement. Iterative method may be useful for huge data. Say data matrix is very big that cannot fit in memory. – hxd1011 May 10 '17 at 18:41

6 @hxd1011 Thank you for clarifying this practical dimension to the problem. I was thinking in purely mathematical terms. – Sycorax May 10 '17 at 18:51

6 Answers

The main reason why gradient descent is used for linear regression is the computational complexity: it's computationally cheaper (faster) to find the solution using the gradient descent in some cases.

The formula which you wrote looks very simple, even computationally, because it only works for univariate case, i.e. when you have only one variable. In the multivariate case, when you have many variables, the formulae is slightly more complicated on paper and requires **much** more calculations when you implement it in software:

$$\beta = (X'X)^{-1}X'Y$$

Here, you need to calculate the matrix $X'X$ then invert it (see note below). It's an expensive calculation. For your reference, the (design) matrix X has K+1 columns where K is the number of predictors and N rows of observations. In a machine learning algorithm you can end up with K>1000 and N>1,000,000. The $X'X$ matrix itself takes a little while to calculate, then you have to invert $K \times K$ matrix - this is expensive.

So, the gradient descent allows to save a lot of time on calculations. Moreover, the way it's done allows for a trivial parallelization, i.e. distributing the calculations across multiple processors or machines. The linear algebra solution can also be parallelized but it's more complicated and still expensive.

Additionally, there are versions of gradient descent when you keep only a piece of your data in memory, lowering the requirements for computer memory. Overall, for extra large problems it's more efficient than linear algebra solution.

This becomes even more important as the dimensionality increases, when you have thousands of

Remark. I was surprised by how much attention is given to the gradient descent in Ng's lectures. He spends nontrivial amount of time talking about it, maybe 20% of entire course. To me it's just an implementation detail, it's how exactly you find the optimum. The key is in formulating the optimization problem, and how exactly you find it is nonessential. I wouldn't worry about it too much. Leave it to computer science people, and focus on what's important to you as a statistician.

Having said this I must qualify by saying that it is indeed **important to understand** the *computational complexity and numerical stability* of the solution algorithms. I still don't think you must know the details of implementation and code of the algorithms. It's not the best use of your time as a statistician usually.

Note 1. I wrote that you have to invert the matrix for didactic purposes and it's not how usually you solve the equation. In practice, the linear algebra problems are solved by using some kind of factorization such as QR, where you don't directly invert the matrix but do some other mathematically equivalent manipulations to get an answer. You do this because matrix inversion is an expensive and numerically unstable operation in many cases.

This brings up another little advantageof the gradient descent algorithm as a side effect: it works even when the design matrix has collinearity issues. The usual linear algebra path would blow up and gradient descent will keep going even for collinear predictors.

edited May 11 '17 at 13:28 answered May 10 '17 at 19:39

Aksakal
37.7k 4 49 112

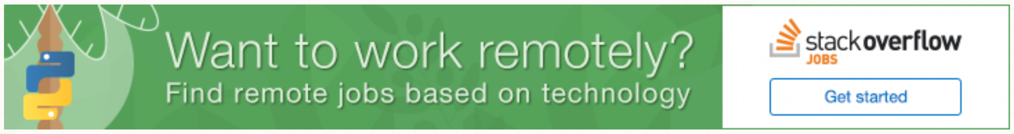
11 But Ng *is* a computer science person. – amoeba May 10 '17 at 19:52

18 Regarding your Remark: As a mathematician, I used to agree. But my understanding now is that in modern machine learning, the method of optimization is inherently tied up with the objective being optimized. Some forms of regularization, like dropout, are more cleanly expressed in terms of the algorithm instead of the objective. In short: if you take a deep net, keep the objective function but change the optimization method, you may get very different performance. In fact, sometimes a better optimizer yields worse results in practice ... – A. Rex May 10 '17 at 21:50

13 Minor nitpick: you'd certainly not *invert* $X'X$; instead you'd *solve* the linear equations system $X'X\beta = X'y$ for β. Abstractly, it's the same, but numerically, it's far more stable, and potentially even cheaper. – Stephan Kolassa May 11 '17 at 8:55

3 @AnderBiguri Solution with a QR factorization, on the other hand, is backward stable, hence it delivers a solution which is as accurate as possible given the uncertainty in the input data. – Federico Poloni May 11 '17 at 12:34

5 I think we should all stop writing $\beta = (X^tX)^{-1}X^ty$ and just write $X^tX\beta = X^ty$ all the time. – Matthew Drury May 11 '17 at 13:32



First, I would strongly recommend to you to read the following two posts (if not duplicate)

Please check J.M.'s answer in

What algorithm is used in linear regression?

Please check Mark's answer (from numerical stability point of view) in

Do we need gradient descent to find the coefficients of a linear regression model

In short, suppose we want to solve linear regression problem with squared loss

$$\text{minimize } \|Ax - b\|^2$$

We can set the derivative $2A^T(Ax - b)$ to 0, and it is solving the linear system

$$A^TAx = A^Tb$$

In high level, there are two ways to solve a linear system. Direct method and iterative method. Note direct method is solving $A^TAx = A^Tb$, and gradient descent (one example iterative method) is directly solving $\text{minimize } \|Ax - b\|^2$.

Comparing to direct methods (Say QR / LU Decomposition). Iterative methods has some advantages when we have large amount of data or the data is very spase.

- Suppose our data matrix A is huge and it is not possible to fit in memory, stochastic gradient decent can be used. I have an answer to explain why How could stochastic gradient descent save

On the other hand, I believe the one of the reason Andrew NG emphasizes it is because it is a generic method (most widely used method in machine learning) and can be used in other models such as logistic regression or neural network.

edited May 11 '17 at 14:58 answered May 10 '17 at 18:48
hxd1011
18k ● 4 ● 48 ● 140

You are absolutely right. SGD is very helpful while handling a big amount of data. The method Prof Ng demonstrating is the most classic and pure one. One should start from that point to have a clear idea. If one can understand the motto of that then whole linear estimation will be crystal clear to him/her. – Sandipan Karmakar May 10 '17 at 19:00

Syncorax is correct that you don't need gradient descent when estimating linear regression. Your course might be using a simple example to teach you gradient descent to preface more complicated versions.

One neat thing I want to add, though, is that there's currently a small research niche involving terminating gradient descent *early* to prevent overfitting of a model.

answered May 10 '17 at 17:59
Tim Atreides
618 ● 1 ● 5

2 For overfitting statement, could you provide the link? is adding the regularization term better than limiting number of the iterations? – hxd1011 May 10 '17 at 19:26

You could look at Chapter 7 of Deep learning by Goodfellow et al , which mentions early stopping to prevent overfitting in neural nets. – Batman May 10 '17 at 20:27

2 Regularization by early stopping is by no means a new technique; it's a well known technique in, say, Landweber iteration: en.wikipedia.org/wiki/Landweber_iteration – cfh May 10 '17 at 20:45

If I am not wrong, I think you are pointing towards the MOOC offered by Prof Andrew Ng. To find the optimal regression coefficients, grossly two methods are available. One is by using Normal Equations i.e. by simply finding out $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$ and the second is by **minimizing the least squares criterion** which is derived from the hypothesis you have cited. By the way, the first method i.e. the Normal equations is a product of the second method i.e. the optimization method.

The method you have mentioned i.e. using correlation, it is applicable for one predictor and one intercept quantity only. Just be noticing the form. So, when the number of predictors is more than one in number then what is the way out? Then one has to resort to the other methods i.e. the normal equation or optimization.

Now why optimization (here Gradient Descent) although direct normal equation is available. Notice that in normal equation one has to invert a matrix. Now inverting a matrix costs $\mathcal{O}(N^3)$ for computation where N is the number of rows in $\mathbf{X}$ matrix i.e. the observations. Moreover, if the $\mathbf{X}$ is ill conditioned then it will create computational errors in estimation. So, it is the Gradient Descent kind of optimization algorithm which can save us from this type of problem. Another problem is overfitting and underfitting in estimation of regression coefficients.

My suggestion to you is don't go for merely solving a problem. Try to understand the theory. Prof Ng is one of the best Professors in this world who kindly teaches Machine Learning in MOOC. So, when he is instructing in this way then it must have some latent intentions. I hope you will not mind for my words.

All the best.

answered May 10 '17 at 18:35
Sandipan Karmakar
381 ● 1 ● 9

5 "Inverting a matrix" is strongly NOT recommended. QR is more numerically stable to solve a linear system. – hxd1011 May 10 '17 at 18:56

1 I agree with the computational argument. However, over- or underfitting has nothing to do with using GD vs. the Normal equation, but rather with the complexity of the (regression) model. Both methods (GD if it works properly) find the same least-squares solution (if it exists), and will therefore over- or under-fit the data by the same amount. – Ruben van Bergen May 10 '17 at 18:56

First, yes, the real reason is the one given by Tim Atreides; this is a pedagogical exercise

However, it is possible, albeit unlikely, that one would want to do a linear regression on, say, several trillion datapoints being streamed in from a network socket. In this case, the naive evaluation of the analytic solution would be infeasible, while some variants of stochastic/adaptive gradient descent would converge to the correct solution with minimal memory overhead.

(one could, for linear regression, reformulate the analytic solution as a recurrence system, but this is not a general technique.)

answered May 10 '17 at 18:46
Timothy Teräväinen
301 1 5

One other reason is that gradient descent is a more general method. For many machine learning problems the cost function is not convex (e.g., matrix factorization, neural networks) so you cannot use a closed form solution. In those cases gradient descent is used to find some good local optimum points. Or if you want to implement an online version than again you have to use a gradient descent based algorithm.

answered Oct 17 '17 at 7:37
Sanyo Mn
322 1 11