Gradient Descent Algorithm: Case Study

Peshawa Jamal Muhammad Ali
peshawa.jammal@koyauniversity.org
Department of Manufacturing Engineering
Faculty of Engineering, Koya University, Koya
KOY45, Kurdistan Region - F.R. Iraq.

Haval Abdulkarim Ahmed
haval.abdulkarim@koyauniversity.org
Department of Software Engineering
Faculty of Engineering, Koya University, Koya
KOY45, Kurdistan Region - F.R. Iraq.

Cite this article:
Peshawa J. Muhammad Ali, Haval Abdulkarim Ahmed; "*Gradient Descent Algorithm: Case Study*", Machine Learning Technical Reports, 2021, 2(1), pp 1-7. DOI: 10.13140/RG.2.2.18026.44489
https://docs.google.com/document/d/1FBgyH6CctC5TMJj_8eQkSxOo_4eOQpl0BkTgZM_5vLA/edit#

Information about the publisher:
Machine Learning Technical Reports is a periodical technical report published by the Machine Learning Laboratory
Koya 44023, Erbil, F.R. of Iraq
Contact: +9647707578801/+9647501138655, email: peshawa.jammal@koyauniversity.org

Abstract
This article aims to clarify the mathematical background of the gradient descent optimization algorithm using a simple case study. Gradient descent is one of the most important optimization strategies used by many machine learning models to find the best variants that minimize the error between predicted functions and empirical data. This algorithm is widely used in deep learning, neural networks, linear regression, logistic regression, etc... There are ready to use gradient descent functions in Matlab, Python, Excel, and Weka that could be used directly, but the mathematical background is still not clear for many of the researchers. This article simply presents how this algorithm is working. At the end of this case study, there is a model function determined that represents the collected data.

Description
To understand the gradient descent it is better to start with a simple real-life case study. Assume a real estate company needs a mathematical model to predict the price of the houses with respect to their area, for simplicity purposes the effect of other features like location of the house in the city, year of construction, number of rooms, and number of floors is not considered, see Table 1.

Table 1: House prices collected from the real estate company

House area (square meter)	House Price (U.S. Dollar)
100	50,000
130	55,000
135	60,000
145	70,000
150	80,000
155	85,000
160	87,500
175	90,000
220	110,000
230	120,000

To visualize the dataset, it is better to use a Microsoft Excel sheet or Google Sheets and plot the points on a cartesian coordinate system, see Fig.1.

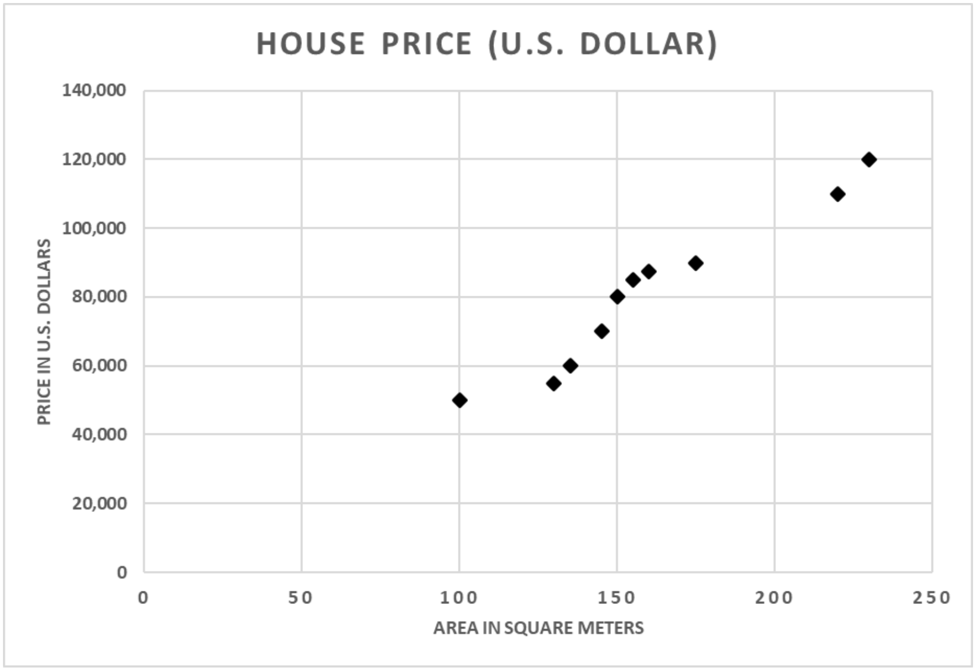

Fig. 1: The collected dataset, area of the houses and their real prices

Click on the link to access the Google sheet:
https://docs.google.com/spreadsheets/d/1oi4Yny9pptDDUUqx6pTOqJBmraHldoF_/edit#gid=168974361

Data Normalization

Before doing any action, it is preferred to normalize the data as it makes the optimization process faster [1]. In our opinion the normalization is required whenever the gradient descent optimization is used, regardless of the fact that the features have similar ranges or not. In this study, the horizontal axis contains values not more than 230 while the vertical axis contains much bigger values like 120000, this difference is one of the reasons that we need to normalize the data. Normalizing means casting or bringing all the values of the dataset to a specific range, according to our experience data normalization is not necessary only in cases if all the features have similar ranges of data which rarely happen in very few types of applications [2][3], because most of the applications containing features have different ranges. It is a good practice to normalize different ranges to a range of [0,1]. A simple formula can be used n'=(x-min)/(max-min). After applying this step the result will be like in Table 2. To visualize the same dataset plotted after being normalized to the range [0,1], see Fig.2.

Table 2: The dataset after normalization to the range of [0,1]

House Area (square meter)	**House Price (U.S. Dollar)**	**Area (normalized) a'=(a-amin)/(amax-amin) a'=(a-100)/(230-100) x**	**Price (normalized) p'=(p-pmin)/(pmax-pmin) p'=(p-50000)/(120000-50000) y**
100	50,000	0	0
130	55,000	0.230769231	0.071428571
135	60,000	0.269230769	0.142857143
145	70,000	0.346153846	0.285714286
150	80,000	0.384615385	0.428571429
155	85,000	0.423076923	0.5
160	87,500	0.461538462	0.535714286
175	90,000	0.576923077	0.571428571
220	110,000	0.923076923	0.857142857
230	120,000	1	1

To represent this data a straight line could be drawn, the general equation of a straight line is (y=mx+c), where m is the slope of the line, and c is intersect with the y-axis, a constant value. In this study, we are looking for the best value of m and c that makes the difference between the empirical and predicted values at the minimum level, therefore, an assumption is made of two random values called initial values.

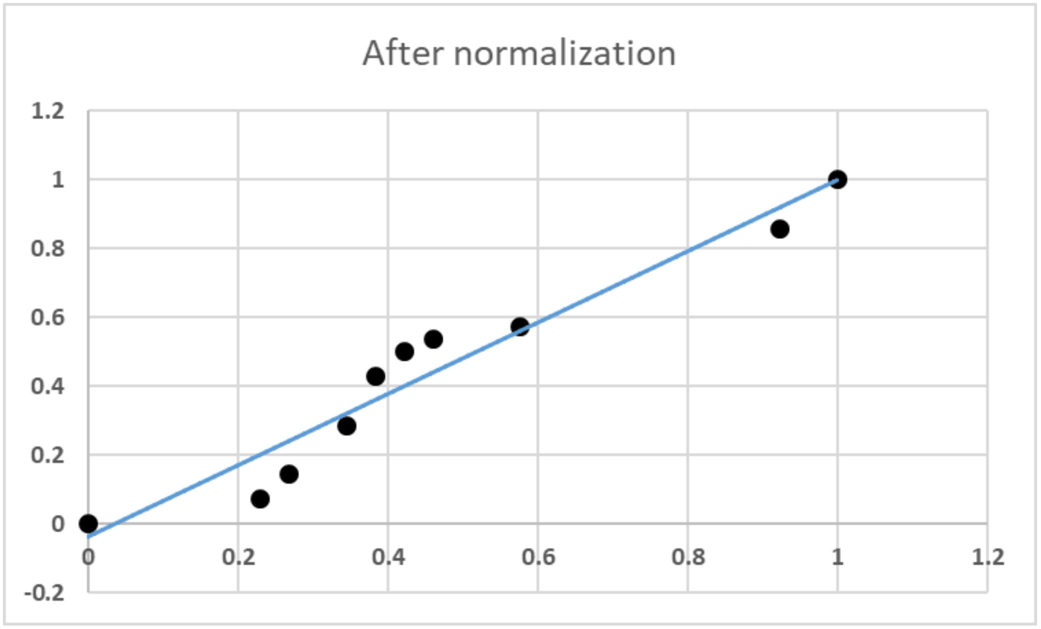

Fig. 2: Plotting points after normalization applied

One of the benefits of the normalization process is that because all the features will be in the same range (like between 0 to 1), the assumed initial value of m and c must be in the same range too or very close to the range, and this will make the optimization process faster.

Gradient Descent

Let us start with initial values like m=1.0, and c = 0.01. Table 3 shows the yp values (i.e. using the formula (y = 1.0 x + 0.01) to predict values of yp using the same x values of the table), then calculate the squared difference between predicted yp and y, see Table 3.

The sum of square error (SSE) = (0.068192066). The sum is always multiplied by (1/2) for simplicity, there will be a kind of reduction and elimination in the next steps, therefore, this (1/2) is added in this step. The SSE will be (0.034096033).

Now, it is time to calculate the partial derivative of the function of SSE=0.5(y-yp)^2 with respect to m and c. This step requires a little knowledge in calculus.

SSE=0.5(y-(mx+c))^2

∂ SSE / ∂c =0.5*2 (y-mx-c)(-1) = -(y-mx-c) = -(y-(mx+c))= – (y-yp)

∂ SSE / ∂m = 0.5*2 (y-mx-c)(-x)= -(y-(mx+c))x = – (y-yp)x

Table 4 shows the partial derivative values with respect to m and c.

Table 3: Calculating y-predict (yp) and squared error

x	y	yp = 1.0*x+0.01	Squared Error (y-yp)^2
0	0	0.01	0.0001
0.230769	0.071429	0.24076923	0.028676259
0.269231	0.142857	0.27923077	0.018597766
0.346154	0.285714	0.35615385	0.004961732
0.384615	0.428571	0.39461538	0.001153013
0.423077	0.5	0.43307692	0.004478698
0.461538	0.535714	0.47153846	0.004118536
0.576923	0.571429	0.58692308	0.00024008
0.923077	0.857143	0.93307692	0.005765982
1	1	1.01	0.0001

Table 4: Calculating partial derivatives of m and c

x	y	yp = m*x+c	Squared Error (y-yp)^2	∂ SSE / ∂c = – (y-yp)	∂ SSE / ∂m = – (y-yp)x
0	0	0.01	0.0001	0.01	0
0.230769	0.071429	0.24076923	0.028676259	0.169340659	0.039078614
0.269231	0.142857	0.27923077	0.018597766	0.136373626	0.036715976
0.346154	0.285714	0.35615385	0.004961732	0.07043956	0.024382925
0.384615	0.428571	0.39461538	0.001153013	-0.033956044	-0.013060017
0.423077	0.5	0.43307692	0.004478698	-0.066923077	-0.028313609
0.461538	0.535714	0.47153846	0.004118536	-0.064175824	-0.029619611
0.576923	0.571429	0.58692308	0.00024008	0.015494505	0.008939138
0.923077	0.857143	0.93307692	0.005765982	0.075934066	0.070092984
1	1	1.01	0.0001	0.01	0.01
		Total	0.068192066	0.322527473	0.118216399
		Total*0.5	0.034096033		

Variant Updating

This step is to update the value of the two important variants m and c according to the rules below:

New m = m – r * ∂SSE/∂m = 1.0 - 0.15*(0.118216399) = 0.98226754

New c = c – r * ∂SSE/∂c = 0.01 - 0.15*(0.322527473) = -0.038379121

Where r is the learning rate = 0.15

Small r value results in slow learning (reaching optimum slower), but no stucking or going in a wrong direction, while big value of r leads to reaching the optimum faster but with a possibility of getting stuck in local minimums or going towards wrong directions. The above process is summarized in this shared sheet:
https://docs.google.com/spreadsheets/d/1oi4Yny9pptDDUUqx6pTOqJBmraHIdoF_/edit#gid=382771220

Each sheet of this Google Sheet contains one trial, sheets are all connected (i.e. sheet no.3 uses the result of sheet no.2, and sheet no.2 uses the result of sheet no.1, and so on.). Sheet no.10 shows the result after 10 trials.

This process will be repeated with the two new values of m and c until reaching a level where calculations give a stable SSE (no significant change in two subsequent steps). In this study, after 20 trials we reached a stable SSE, but we continued to 50 trials to get better stability (m=1.0365, and c=-0.0391), therefore, the resultant equation will be:

$$y=1.0365\,x - 0.0391$$

Denormalization

After getting the best values for variants m and c, we must return back the resultant equation to values representing the real life problem actual data. Three equations must be replaced into each other:

p = [p' *(pmax-pmin)]+pmin, where p, p', pmin, and pmax are prices

y=1.0365 x - 0.0391

a'=(a-amin)/(amax-amin), where a', a, amin, and amax are areas

Therefore,

p = [p' *(120000-50000)] + 50000	the denormalized equation of prices
p = [(1.0365 x -0.0391)*(70000)] + 50000	instituting y instead of p'
p = [72555 x - 2737] + 50000	
p = [(72555 *(a -amin)/(amax-amin)) - 2737] + 50000	instituting a' instead x
p = [(72555 *(a -100)/(230-100)) - 2737] + 50000	
p = [(72555 *(a -100)/130) - 2737] + 50000	
p = [558.1153846153846 * a - 55811.53846153846 - 2737] + 50000	

p = 558.115383 *a - 8548.538461

For any new instance like a house of an area equal to 200 square meters, then the predicted price by the model will be 103,075 U.S. Dollars.

Conclusion

Gradient descent is one of the efficient algorithms that can be used in finding variant optimum values of the modeling equations. Normalization process will speed up the repetitive steps of reaching the optimum solution, the resultant modeling equation needs to be denormalized at the end of the process to represent the real life problems directly.

References

1. Peshawa J. Muhammad Ali, Rezhna H. Faraj; “Data Normalization and Standardization: A Technical Report”, Machine Learning Technical Reports, 2014, 1(1), pp 1-6. DOI:10.13140/RG.2.2.28948.04489 https://docs.google.com/document/d/1x0A1nUz1WWtMCZb5oVzF0SVMY7a_58KQuIqQVT8LaVA/edit#
2. Peshawa J. Muhammad Ali, Nigar M. Shafiq Surameery, Abdul-Rahman Mawlood Yunis, Ladeh Sardar Abdulrahman; “Gender Prediction of Journalists from Writing Style”, ARO-The Scientific Journal of Koya University, 2013, 1(1), pp 22-28. DOI: https://doi.org/10.14500/aro.10031
3. Peshawa J. Muhammad Ali; “Predicting the Gender of the Kurdish Writers in Facebook”; Sulaimani Journal for Engineering Sciences, 2014, 1(1), pp 18-28. https://www.iasj.net/iasj/download/cefc94145e4e8d17