Lecture 3 - Assignment 04
Gradient Descent

EECE-6822 Machine Learning

Cris Ababei
Electrical and Computer Engr., Marquette University

1. Objective

The objectives of this activity include: (1) run several given code examples that deal with gradient descent (GD) in order to get better insights; and (2) conduct an investigation on the computational runtimes/speed of three different GD approaches.

2. Prerequisite Readings

Murphy

- Gradient descent: Murphy 8-8.2.1

Geron: Ch. 4 (second part, starting with Gradient Descent and up to Polynomial Regression)

- Discusses gradient descent (GD) idea. It works well because MSE cost function is convex.
- Focuses on Batch GD (should be called full GD because it uses the whole batch of data at every step); presents briefly equations of gradients, which are used in a simple example code. Note result obtained is exactly like that obtained with the code that implemented the Normal Equation!
- Discusses Stochastic GD (parallel to simulated annealing made); presents simple code that implements it. Presents also Mini-batch GD.
- Polynomial Regression
- Important takeaway: when using GD, you should ensure that all input features have a similar scale! (using SciKit-Learn SnadardScaler class) or it will take much longer to converge.

Raschka – Ch.2

- A general discussion of ML algorithms used for several supervised learning algorithms for classification. Focuses on the use of SciKit-Learn library, which offers a user friendly and consistent interface for using these algorithms. Discussion of strengths and weaknesses of classifiers.

3. Examples

Example 1: Gradient Descent - SciKit-Learn

In this activity, we use code examples from:
[*B3-Geron] Aurelien Geron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, O'Reilly, 2022.

We continue to look at the discussion from Chapter 4. This example builds on the discussion before in this chapter; it looks at a very simple polynomial example **y=4+3*X +np.random.randn(m,1)**.
It starts with a discussion of linear regression, solved directly using the normal equation:
theta_best = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ y
and compares that to the use of **scipy.linalg.lstsq()** as well as SciKit-Learn's **LinearRegression** class. Note the use of **@** for matrix multiplication, as opposed to np.matmul(A,B).
--It then continues presenting **batch gradient descent (GD)**; implemented directly using the gradient of the MSE cost/loss function to iteratively finding theta_best, starting from initial randomly generated values.
--It presents the **stochastic gradient descent (SGD);** it talks about learning schedule and presents a direct implementation, which then is compared to SciKit-Learn's **SGDRegressor** class.

--Finally, it discusses **mini-batch gradient descent**.

The corresponding notebook is at (the first part of it on regression, we did in the previous lectures):
https://github.com/ageron/handson-ml3/blob/main/04_training_linear_models.ipynb
Launch that in your google colab and go through the steps of the **Gradient Descent** Section. Of course, go through any previous portion of the notebook that this one depends on. Also, read the book; more specifically, you must run the example code from pp. 143 (Batch GD) and from pp. 146 (Stochastic GD).

Example 2: Gradient Descent - Python Implementations

This activity uses code example from:
[*B3-Raschka] Sebastian Raschka, Yuxi Liu, and Vahid Mirjalili, Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python, Packt Publishing, 2022.

We will continue or re-do the code examples from Chapter 2.
The first part of this chapter presents the class based **implementation in Python** of the single perceptron used on the **Iris dataset** for binary classification.
Despite the fact that this chapter introduces and talks about classification (we will cover classification later in this course), it introduces minimizing loss functions with gradient descent (pp. 37). This is done in the context of presenting another type of single-layered **neural network (NN)**: **ADAptive LInear NEuron (Adaline)**. In Adaline, the weights are updated based on a linear activation function rather than a unit step function like in the perceptron (presented in the first part of this chapter); the activation function is simply the identity function of the input!

Take time and study the implementation of Adaline in Python with class AdalineGD. Read the entire chapter 2.
Also, look at feature scaling employed to improve gradient descent!
Finally, study the implementation of stochastic gradient descent (SGD).

The examples in this chapter are developed using an object-oriented approach, which will help with understanding the scikit-learn API, that is implemented based on the same core concepts: fit and predict methods.

Finally, in addition to the code examples from this chapter, you can look again briefly at the code examples from Ch.9 (discussed in our previous lectures on linear regression) – where regression parameters are solved also using gradient descent – see pp. 278.

Example 3: Gradient Descent - mlm tutorials and code examples

In this part, we will look at several tutorials from **machinelearningmastery (mlm)**. Read the following tutorials and run the code where applicable:

How to Implement Gradient Descent Optimization from Scratch
https://machinelearningmastery.com/gradient-descent-optimization-from-scratch/

Mini-Batch Gradient Descent and DataLoader in PyTorch
https://machinelearningmastery.com/mini-batch-gradient-descent-and-dataloader-in-pytorch/

4. Assignment

The objective is to apply (1) batch gradient descent, (2) stochastic gradient descent (SGD), and (3) mini-batch gradient descent from Example #1 to the Ames Iowa Housing Prices Dataset which you investigated in the previous Assignment 3. You will have to report and compare the parameters found with each of the three GD versions and the MSE of each GD trained model when tested on the test portion of the dataset. A second objective is to compare their execution times.

--(a)Create a new Jupyter notebook (or work directly as a Python program) that starts with the code that imports the dataset and keeps only the two most important features that you found in the previous assignment. Here, you are asked to use only the best 2 (not 3) out of the 23 input features, in order to make the problem and coding a bit easier. You will again split the dataset into 80/20 train/test portions.

--(b)Use code from the **Gradient Descent** Section of the example done in Example #1 above. You will only need the code that implements: Batch GD (page 143 in the actual book), SGD (page 146 in the book), and mini-batch GD (code not listed in the book, but available in the Jupiter notebook itself: https://github.com/ageron/handson-ml3/blob/main/04_training_linear_models.ipynb
You do not need the code that create the figures from this Section; you can eliminate the code for plotting the figures.

--(c)Employ a polynomial regression model, with a degree "p" that you found as the **optimal effective complexity** in the previous Assignment 3. Add also code before and after each of the three GD implementations that you will run to be able to measure and calculate the execution times. For example, you can use:

start = time.process_time()

end = time.process_time()

and then take the difference. In all three runs use **n_epochs = 100**.

--(d)For each of the above 3 simulations collect the final values for "theta". And report them in a Table. Also, calculate the MSE achieved on the test portion of the dataset. Collect all 3 simulation runtimes and plot them in a graph.

--(e)Present a discussion of your findings.

5. Deliverables

You must write (typed) a report and upload it as a PDF file on D2L. The report should be named "**hw4_report_LastName.pdf**". You should also create a .zip archive with all your code and implementations of all parts of the assignment. Upload also this archive .zip file with the name "**hw4_implementation_code_LastName.zip**" to D2L. Hence, your D2L should contain two items: the report and the .zip file. **Do not include the report inside the .zip and upload only the .zip. They should be two separate items!**

The report should include the following sections and subsections. Make sure section titles are in bold font and pages are numbered.

1) **Title + course info + your name**
2) **Summary.** Describe in one paragraph what the objective of the assignment is.

3) **Gradient Descent Methods.** Write a paragraph to describe the main differences between the three types of GD methods.
4) **Description of Experiments and Discussion.** Describe the experiments you did. All tables and figures should be numbered and should have captions. All plots in all figures should have axes labels and titles. Present a meaningful discussion with the interpretation of the results you obtained. Explain if you expected the results or not; discuss the intuition behind it.
5) **Conclusion.** Present your conclusions; highlight what are your main takeaways that you learned from this assignment. Describe what issues you encountered and how you solved them.
6) **References.** Include all references that you used, as a numbered list. Cite them in the report itself; do not just list them! If your report has References that are not cited in the report, points will be deducted!