Lab 08 – Optimization Methods: Gradient Descent
CS3400 Machine Learning

Learning Outcomes

- Implement an iterative numerical optimization algorithm to solve for model variables (scalar and vector forms)
- Develop the skill of applying optimization methods for solving cost functions
- Explore the implications of global versus local optima, and how the initial conditions affect the result

Overview

Optimization problems can be broken into two major components:

1. A **cost function** that describes the error between the model predictions and given data. This output of this *function* is dependent on the choice of model parameters and the provided training data.
2. An **algorithm** that adjusts parameters to either minimize (or maximize) a cost function. In our applications, we typically try to *minimize* the cost function (model error).

This lab will focus on the **algorithm** step (#2) through coding/implementation of a numerical gradient descent solver. Additionally, you will perform experiments where you apply this algorithm and explore some implications of solving optimization problems, starting parameter values, and global versus local optima. Gradient descent is an iterative algorithm and is defined by the following steps:

1. Set an appropriate cost function for the problem that you are solving.
    a. Cost functions like those created in lab 06.
    b. Could be any function with adjustable parameters.
2. Make an initial guess of the model parameters, $\vec{x}_0$.
3. For k iterations:
    a. Compute the gradient, $\nabla f(\vec{x})$, at current model parameters.
    b. Choose a step size γ[†]
    c. Update $\vec{x}_{k+1} = \vec{x}_k - \gamma \nabla f(\vec{x})$
    d. If change in gradient is < tolerance[⋆] stop – else go to step 3.

[†] for this lab we will use a constant step size for all iterations. What benefit/advantage could an adjustable step size achieve?

[⋆] for this lab the change in gradient can be measured using distance between the current parameter vector ($\vec{x}_k$) and previous parameter vector ($\overrightarrow{x_{k-1}}$).

Instructions

You have been provided with two main .py files (optim_stub.py, test_optim.py). The file optim_stub.py contains a partial implementation of the gradient descent algorithm. Your job is to finish the implementation. The file test_optim.py contains unit tests designed to help you check the correctness of each method.

A Word on Objective / Cost Functions

With gradient descent implemented you will perform several experiments where you will optimize objective functions. For our purposes we will use objective functions to refer to a general form of functions that we are trying to *optimize* or find the extrema of. We can then define cost/loss functions as functions that incorporate both a model (for making model predictions) and an error metric (like MSE). To this point, the third experiment directly uses your previously coded gaussian distribution cost function. However, the first two experiments will take a simpler approach with the objective function. Experiment 1 and 2 provide more straight-forward objective functions in a much simpler form. For these experiments you will be using your implemented algorithm to find the value of x that minimizes the function - f(x). While the functions themselves are trivial (if you plot them – I am sure you can find the answer), they will allow you to better explore what the optimization algorithm is doing.

Optimizer API

Take a moment to look through the optim_stub.py file that you were provided. Each of the methods provided fill their own roles. Please make sure that you are adhering to the required method arguments.

- __init__(self, step_size, max_iter, tol, delta)
- optimize(self, cost_func, starting_params)
- _calculate_change(self, old, new)
- _gradient(self, cost_func, params)
- _update(self, param, gradient)

From these methods you may notice several things.

- Underscores are still used to denote private methods from public functions. For this API, the only public facing method will be *optimize*.
- The class is instantiated with arguments for the optimizer's hyperparameters of step size, max iterations, tolerance, and delta. These hyperparameters are constant for the instantiated object.
- *_calculate_change, _gradient*, and *_update* are helper methods for the optimizer and will therefore be called in the *optimize* method.
- If you successfully completed Lab 07, the *_gradient* method should be 99% complete. If you were not able to get a working solution, talk with you professor as it is required for this lab.

- The cost function is passed as an argument to the optimize method. Your optimizer should eventually reference the cost_func.cost method. This also means that if you are using an objective function, you may choose to create a new class for each objective function that has a single cost method that returns the expressions value.

Jupyter Notebook – Running experiments

Create a Jupyter notebook named <lastname>_lab08. The notebook should have a title, your name, and an introduction. You will be running the following experiments in the notebook.

Each of the following steps should be performed in individual cells in your jupyter notebook. If you wish to write a .py script to perform some of these operations and import it into your notebook, that is fine, but the individual segments should be in their own cells.

Experiment 1: Cubic Model

Experiment 1 will use a cubic expression as an objective function. For this lab you will first plot this function over a few ranges to get a feel for what the function looks like. For our purposes this plot that you create can be considered the "objective function space". It is an optimizer's purpose in life to use tools from calculus (derivatives) to locate the optima (or extrema). However, in machine learning you often won't have a good way to visualize this space.

Cubic Model:

$$f(x) = x^3-3x^2-144x+432$$

Remember that the f(x) can be interpreted the same as the error metric (like MSE) from your cost functions implemented in lab06.

1. In the notebook, plot the cost function above within the bounds of $-15 \leq x \leq 18$ with a step size of 0.01. Observe the shape of this function and at what values of x the function has extrema.
2. In a new figure, plot the numerical derivative of the cubic model within the same bounds and using the same step size. Search through the numerical derivative for the first 2 points of the that are equal (or closest) to zero.
3. In a new cell in your notebook create a new objective function class. This class should implement the given objective function in a method called *cost*. Hint: Follow the cost_function API – but you will only need to implement the *cost* method.
4. Import your optim classes, create an object with it, and supply the cost function that you wrote. Use the following *hyperparameters*
   - Step_size = 0.01
   - max_iter = 100
   - tol = 1e-5
   - delta = 1e-4
5. Using a starting point of 0, solve for the minimum using your optimizer. Plot a line from this starting point to the minima on your objective function plot.

Experiment 2: Quartic Model

Quartic Model:

$$f(x) = 3x^4-16x^3-18x^2$$

1. In the notebook, plot the cost function above within the bounds of $-3 \leq x \leq 6.75$ with a step size of 0.1. Observe the shape of this function and at what values of x the function has extrema.
2. In a new figure, plot the numerical derivative of the quartic model within the same bounds and using the same step size. Solve for the first 3 points of the derivative that are equal (or closest) to zero.
3. In a new cell in your notebook create a new objective function class. This class should implement the given objective function in a method called *cost*.
4. Instantiate a new object with your optim class and supply the cost function that you wrote. Use the following *hyperparameters*
   - Step_size = 0.001
   - max_iter = 1000
   - tol = 1e-5
   - delta = 1e-4
5. Using the following three starting points, solve for the minimum using your optimizer. Plot a line from each of the starting points to the found minima on your objective function plot.

Starting Point
x = 6
x = 3
x = -2

Experiment 3: Gaussian Model

In this experiment you will use the optimizer to finally solve for the correct μ and σ parameters that *minimize* the error between your implemented gaussian distribution model and the gaussdist.csv dataset you encountered in labs 2, 6, and 7.

- Import the gaussian distribution cost function that you created in the previous labs and instantiate it with the gaussdist.csv dataset.
- Follow the presentation format from the previous labs in this notebook. Make sure to plot the data, initial model predictions, and the solved model predictions. Make sure to include in the plot the original error and the error after using the optimizer.
- Use a starting point (initial prediction) of μ = 1 and σ = 0.75.
- Use the following *hyperparameters*
    - Step_size = 1
    - max_iter = 5000
    - tol = 1e-4
    - delta = 1e-3

Questions:

After you run all the experiments create a markdown cell to answer questions in. Copy and paste each question into the cell prior to answering it. The following questions will be graded:

1. Reflect on the form and organization of our optimizer API. Specifically, discuss each of the methods and what role they serve. This discussion should include what arguments they accept, what the method returns, and why we might choose to separate out these specific methods into helper methods.

2. For experiment 1:
    i. how many optima did you find? Hint: Discuss the significance of places where the derivative is equal to 0.
    ii. When you used the optimizer you started at x = 0. How many optima did your optimizer return? Was it a minima or maxima? Was it a global or local optima? By looking at the gradient descent algorithm find what term pointed you toward the minimum. Describe how it did this. Can you think of a way to find the function's maxima?

3. For experiment 2:
    i. how many optima did you find?
    ii. Describe the different starting locations that you used to solve for optima. Was the found optima different for any of these starting locations and were they the

global or local optima? If it was, can you explain why the optimizer found different solutions?

4. For experiment 3:
    i. how many optima did you find?
    ii. Look back at the heatmaps you generated in Lab06 for the gaussian distribution. Describe what the optimizer is doing using the heatmap visualization.

I will be will looking for the following:

- That all of the tests pass
- That you used Numpy functions as much as possible and avoided writing loops where possible
- You avoided unnecessary calculations and memory copies where reasonable
- An introduction (including your own statement of the problem/lab purpose) and a written summary of your results at the top of the notebook in Markdown. Make sure to put your name at the top of the notebook.
- Plots of your data look reasonable. Plots have labels for each axis.
- Reasonable answers to the questions.

Submission Instructions

Save the notebook as a pdf file named <lastname>_lab08.pdf. Put your pdf file, cost_functions.py file, and optimp.py file into a zip file. Upload the zip file to Canvas.

Rubric

Presentation	**10%**
Followed submission and formatting instructions	5%
Plots: axes are properly labeled, used correct axes for variables, points were colored as required, lines were coloring as required, used a legend, chose appropriate axes limits to make plot readable and do not cause misleading interpretations, etc.	5%
Optim Function Coding	**25%**
Implementation passes unit tests	20%
Used Numpy functions wherever possible (no Python loops)	5%
Experiment #1	**15%**
Cost Function	5%
Correct Plots – includes all requested figures and data points	5%
Optimizer is used correctly to find optima	5%
Experiment #2	**15%**
Cost Function	5%
Correct Plots – includes all requested figures and data points	5%
Optimizer is used correctly to find optima	5%
Experiment #3	**10%**
Correct Plots – includes all requested figures and data points	5%
Optimizer is used correctly to find optima	5%
Reflection Questions	**20%**
Problem 1	5%
Problem 2	5%
Problem 3	5%
Problem 4	5%
Exceeds Expectations	**5%**