E178 SDSE: Lab 2

Solving optimization problems with Gradient Descent and Stochastic Gradient Descent

September 12, 2023

About Me

- Originally from Denver, Colorado

Undergrad

PURDUE MECHANICAL ENGINEERING

- BSME 2019

Grad

Mechanical Engineering
UNIVERSITY OF CALIFORNIA, BERKELEY

- Started August 2019
- Berkeley Fellowship & NSF GFRP
- PI: Dr. Masayoshi Tomizuka

- Chair of conference planning committee:

expanding your horizons AT BERKELEY March 12, 2022

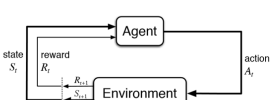
- Instructor/tutor:

SIG SUMMER INSTITUTE FOR THE GIFTED

COMMUNITY RESOURCES FOR SCIENCE

Research

- Controls and machine learning
- Autonomous racing



INTRODUCTION

We will cover:

- Gradient descent (GD) and stochastic gradient descent (SGD)

All of the details will be covered in point estimation lecture. For now, we define the problem:

Problem Definition

Suppose that we have a system that produces real-valued (scalar) measurements. The system is modeled as a random variable Y with an unknown pdf p_Y. The dataset $\mathcal{D}$ consists of N measurements iid sampled from Y.

$$\mathcal{D} = \{y_i\}_N \sim Y$$

Our goal is to use $\mathcal{D}$ to estimate a distribution p that best approximates p_Y.

We will use the *maximum likelihood* technique, which solves the problem in two steps:

1. Guess which family of distributions best fits Y
2. Solve an optimization problem for the optimal values of the parameters

INTRODUCTION

We will use the *maximum likelihood* technique, which solves the problem in two steps:

1. Guess which family of distributions best fits Y
2. Solve an optimization problem for the optimal values of the parameters

For step 1, we will examine two different assumptions:

- Y is exponential distribution
- Y is a normal distribution

To solve step 2:

$$\underset{\theta_1,...\theta_D}{\text{minimize}} \; -\frac{1}{N}\sum_{i=1}^{N} \ln p(y_i; \theta_1, ...\theta_D)$$

- $p(y_i; \theta_1, ...\theta_D)$ is the pdf of the selected family with parameters $\theta_1, ..., \theta_D$, evaluated on $y_i \in \mathcal{D}$
- For now, simply regard this as a function to be minimized $\rightarrow$ refer to as the ***cost function***

$$\mathcal{J}(\theta_1, ...\theta_D; \mathcal{D}) = -\frac{1}{N}\sum_{i=1}^{N} \ln p(y_i; \theta_1, ...\theta_D)$$

Goal: minimize $\mathcal{J}$ over the parameters $\theta_1, ..., \theta_D$ with the dataset $\mathcal{D}$ held fixed.

LOGISTICS

- Lab Groups
  - You should have received an email with your lab group assignment
  - If there are any problems, you must tell me by the end of lab TODAY
  - This will be your lab group for the remainder of the semester
- Turning in the Lab:
  - The report is due by the beginning of your session on the third week, i.e. the first session of the next lab. In the example, if you are in lab section 101, then your lab report for lab 2 is due by Tuesday 9/26 at 10 am.
  - Report submission will be though bCourses.
- Autograder
  - We will use the package `otter-grader` for autograding your coded
  - Your job is to locate the `#TODO` in the code and fill in the appropriate code to complete the question
  - After each `#TODO`, you can run the autograder to check your answer
  - We will do the first question in lab so you can see how this works

Python Libraries and Functions

Dependencies

- `pip install matplotlib`
- `pip install numpy`
- `pip install otter-grader`

Useful Functions

- `import numpy as np`
  - `np.array(x)`: makes an array from a list or list of lists
  - `np.mean(x)`: takes the mean of `x`, see documentation for more options such as `axis`
  - `np.log(x)`: takes the natural log of `x`
- `import matplotlib.pyplot as plt`
  - This is a plotting library that is meant to be very similar to the MATLAB plotting functions
  - e.g. `plt.plot(x,y)`

NUMPY DIFFERENCES TO MATLAB

Numpy handles arrays in a very similar way to MATLAB.

However there are a few key differences:

- List Comprehension: numpy arrays are indexed starting at 0! This is different that MATLAB which indexes starting at 1.
- Element-wise vs. Matrix multiplication
  - e.g. `A*B` is element-wise multiplication - analogous to `A.*B` in MATLAB
  - e.g. `A@B` is matrix multiplication - analogous to `A*B` in MATLAB

GETTING STARTED

1. Open `lab2.ipynb` in Visual Studio
2. If you have not already: create a virtual environment with

```
python -m venv sdse_env
```

   - This creates a new virtual environment named `sdse_env`
3. Run

```
pip install -r requirements.txt
```

4. Run the first cell $\rightarrow$ this initializes the autograder
   - You should not see any error - if you do, raise your hand
5. Run **EVERY** cell and read the description of what each cell is doing
6. Your job is to replace the `#todo` with the appropriate code based on the description in the notebook
7. After each `#todo`, run the next cell to run the autograder and check if you implemented it correctly
8. We will do 1.2 through 2.1 together