Lecture 9

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Introduction

- We will look at continuous optimization concepts in this lecture.
- There are two main branches of continuous optimization - constrained and unconstrained optimization.
- We seek the minimum of an objective function which we assume is differentiable.
- We study the Gradient Descent Algorithm to optimize the objective function without constraint

Motivation

Consider the data in the given table

x	y
1	3.1
2	4.9
3	7.3
4	9.1

- Easiest model of y one can think of is $y = ax + b$. Let $\hat{y} = ax + b$ be the y predicted.
- Our aim is to find a and b such that the difference between y and the predicted $\hat{y}$ is minimum. The loss function can be defined as $L(a, b) = \sum_{i=1}^{4}(y_i - (ax_i + b))^2$. Then the problem will be to find a and b such that $L(a, b)$ is minimized which is an optimization problem.

Example

- Suppose we want to find local extremum values of the function $l(x) = x^4 + 7x^3 + 5x^2 - 17x + 3$.
- Obviously this is an unconstrained optimization problem and we want to use the calculus method to find the extremum values.

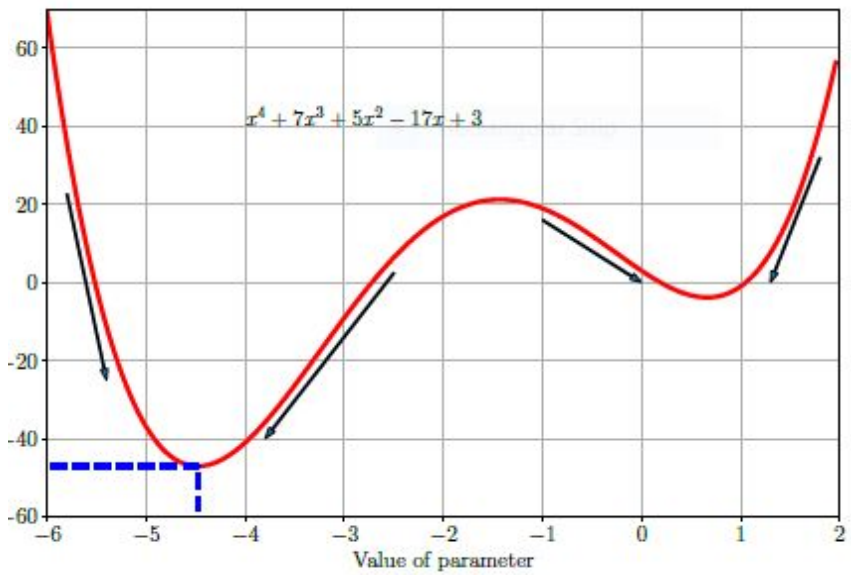
  - The gradient is $\frac{dl(x)}{dx} = 4x^3 + 21x^2 + 10x - 17$.
  - Setting the gradient to zero identifies points corresponding to a local minimum or local maximum - there are three such points since this is a cubic equation.
  - The second derivative is $12x^2 + 42x + 10$

Optimization using gradient descent

- For low-order polynomials we can solve the equations analytically and find points at which the gradient is zero.
- Consider the problem of solving for the minimum of a real-valued function $\min_x f(x)$ where $f : R^d \rightarrow R$ is an objective loss function.
- We assume our function f is differentiable but that the minimum cannot be found analytically in closed form.
- The main idea of gradient-descent is to take a step from the current point of magnitude proportional to the negative gradient of the function at the current point.
- This is explained pictorially in the following slide.

The directions of movement in the Gradient Descent Algorithm

- We move in the direction of the negative gradient to decrease the objective function.
- We move until we encounter a point at which the gradient is zero.



Gradient Descent Algorithm

- start at some initial point x_0 and then iterate according to

  $$x_{i+1} = x_i - \alpha((\nabla f)(x_i))^T$$

- For a suitable step-size α, the sequence of points $f(x_0) \geq f(x_1) \geq \ldots$ converges to some local minimum.
- The parameter α is called learning-rate

Pseudo code for Gradient Descent Algorithm

Assume $\theta \in R^D$, and that we know both $f(\theta)$ and its gradient $\nabla f(\theta)$.
Here is pseudo-code for gradient descent on an arbitrary function f.
The parameter α is often called learning rate when gradient descent is applied in machine learning.
ε is a very small number which one has to fix .
θ_{init} is the initial value of θ

Gradient Descent(θ_{init} , f, ∇f, ε) {
$\theta_0 = \theta_{init}$
k = 0
Repeat{
k = k+1
$\theta_k = \theta_{k-1} - \alpha \nabla f(\theta_{k-1})$
Until $| f(\theta_k) - f(\theta_{k-1}) | < \varepsilon$ }

Return θ_k }

Example: Applying Gradient Descent Algorithm to Linear Regression

Let $(x_1, y_1), (x_2, y_2) \ldots, (x_N, y_N)$ be given data points. We want to find $w = (w_0, w_1)$ such that the loss function

$E(w) = \frac{1}{2}\sum_{i=1}^{N}[\,(w_0 + w_1 x_i) - y_i]^2$ is minimum.

Initialize $(w_0{}^1, w_1{}^1)$.

K =1

Repeat until convergence{

$$w_0{}^{k+1} = w_0{}^k - \alpha \frac{\partial}{\partial w_0}\left[\frac{1}{2}\sum_{i=1}^{N}[\,(w_0 + w_1 x_i) - y_i]^2\right]_{(w_0{}^k, w_1{}^k)}$$

$$= w_0{}^k - \alpha\left[\sum_{i=1}^{N}\left[\,\left(w_0{}^k + w_1{}^k x_i\right) - y_i\right]\right]$$

$$w_1{}^{k+1} = w_1{}^k - \alpha \frac{\partial}{\partial w_1}\left[\frac{1}{2}\sum_{i=1}^{N}[\,(w_0 + w_1 x_i) - y_i]^2\right]_{(w_0{}^k, w_1{}^k)}$$

$$= w_0{}^k - \alpha\left[\sum_{i=1}^{N}\left[\,\left(w_0{}^k + w_1{}^k x_i\right) - y_i\right]x_i\right]$$

k = k+1 }

return $(w_0{}^{m+1}, w_1{}^{m+1})$ [Assume convergence takes place at the mth iteration]

Questions:

Q1. Why each iteration $\theta_i = \theta_{i-1} - \alpha \nabla f(\theta_{i-1})$ takes you towards the local minimum? [Answered in the following two slides]

Q2. How do you initialise x_0? We can initialize x_0 with any random value. For non-convex optimization problem initialization is challenging. But that is beyond the scope of our course. If you have further interest refer https://arxiv.org/abs/2012.12141.

Q3. How do you fix the learning rate α ? We will to answer this question in little deeper.

Answer to Q1

Suppose $f(x, y, z)$ is differentiable function $\mathbf{u} = u_1\boldsymbol{i} + u_2\boldsymbol{j} + u_3\boldsymbol{k}$ a unit vector, then as we know the gradient of f =

$$\nabla f = \frac{\partial f}{\partial x} i + \frac{\partial f}{\partial y} j + \frac{\partial f}{\partial z} k$$

and the directional derivative of f along the unit direction **u** is given by

$$D_u f = \frac{\partial f}{\partial x} u_1 + \frac{\partial f}{\partial y} u_2 + \frac{\partial f}{\partial z} u_3$$

We ask the question along which directions increasing rapidly and which direction decreases rapidly? The following slide answers this question.

Answer to Q1

1. The function f increases most rapidly when $\cos\theta = 1$ or when $\theta = 0$ and **u** is the direction of ∇f. That is, at each point P in its domain, f increases most rapidly in the direction of the gradient vector ∇f at P. The derivative in this direction is

$$D_u f = |\nabla f| \cos(0) = |\nabla f|$$

2. Similarly, f decreases most rapidly in the direction of - ∇f. The derivative in this direction is

$$D_u f = |\nabla f| \cos(\pi) = -|\nabla f|$$

3. Any direction **u** orthogonal to a gradient $\nabla f \neq 0$ is a direction of zero change in f because θ then equals $\pi/2$ and

$$D_u f = |\nabla f| \cos(\pi/2) = |\nabla f|.0 = 0$$

Large and Small Learning Rates

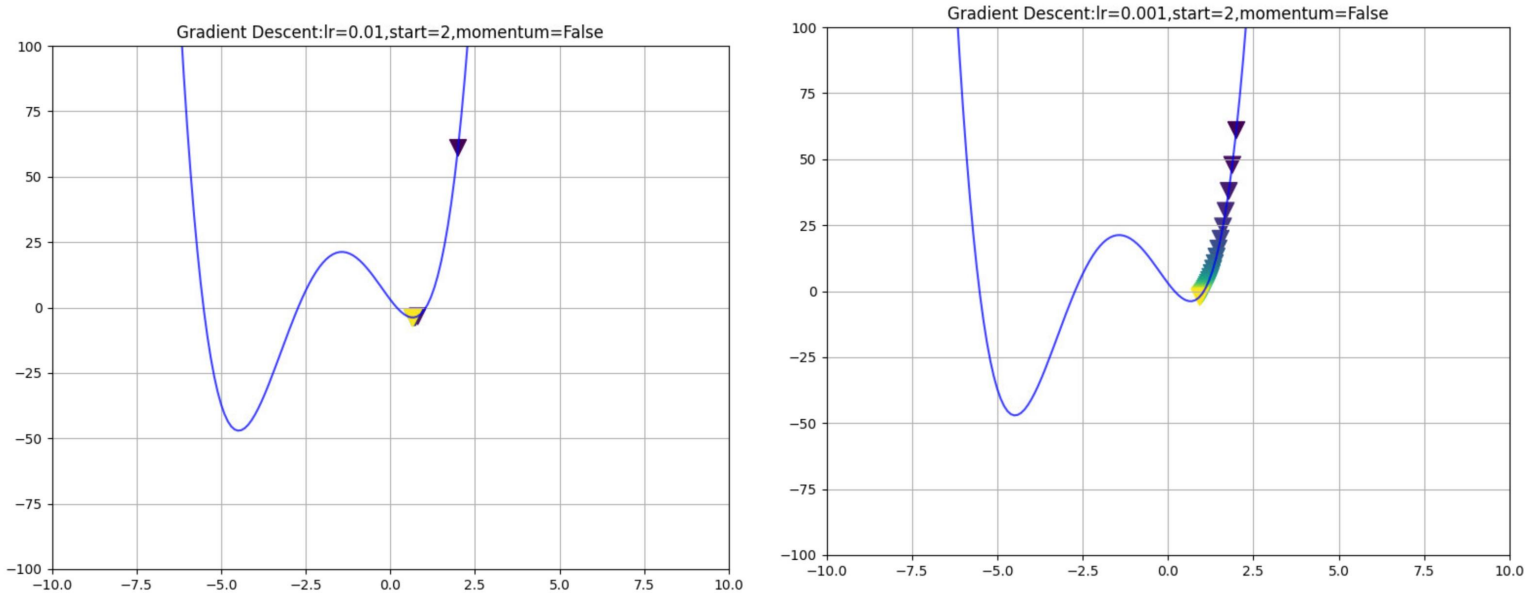


Figure: **left**: with a learning rate of 0.01, local minimum is reached within a couple of steps. **right**: When learning rate is reduced to 0.001, we need relatively more steps to reach the local minimum

Exampl
e

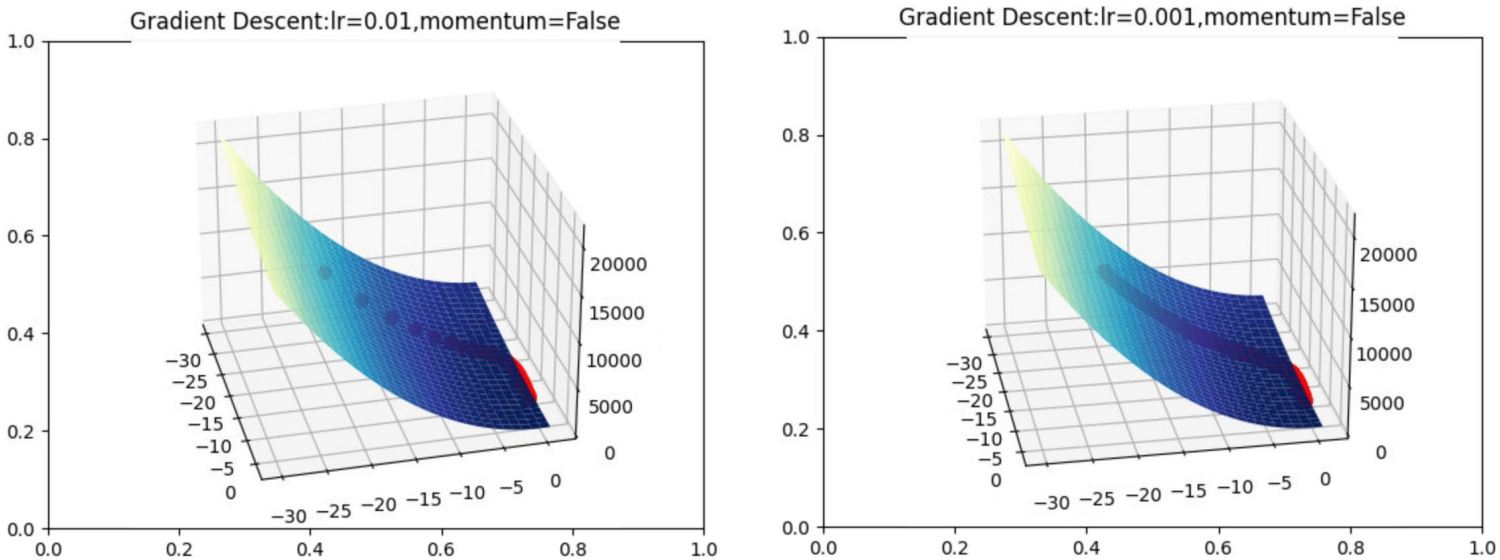


Figure: **left**: with a learning rate of 0.01, minimum is reached. **right**: When learning rate is reduced to 0.001, we need relatively more steps to reach the minimum

Batch gradient descent

- Consider a machine learning problem consisting of loss functions incurred at N data points.
- Let the loss function at the i th data point be $L_i(\theta)$
- Total loss $L(\theta) \quad \sum_{i=1}^{i=N} L_i(\theta)$.
- Here θ is the parameter vector of interest
- The standard gradient descent procedure is a batch optimization method $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha \sum_{i=1}^{i=N} \nabla L_i(\boldsymbol{\theta}^T)$.

Finding Gradient of L at all data points (When the number of data points is very large) is time consuming.

So we follow various strategies to address this issue.

Mini-batch stochastic gradient

- Let S be a subset of the indices $\{1, 2, \ldots N\}$.
- The set S of data points can be treated as a sample and a sample-centric objective function can be constructed as follows:
- $$L_S(\theta) = \sum_{i \in S} L_i(\theta)$$
- The update equation in case of mini-batch stochastic gradient descent can be written as

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha \sum_{i \in S} \nabla \boldsymbol{L}_i(\boldsymbol{\theta}_i)^T$$

- This approach is referred to as mini-batch stochastic gradient.

Stochastic Gradient Descent

- In the extreme case S can contain only one index chosen at random, and the approach is then called as stochastic gradient descent.
- The key idea in stochastic gradient descent is that the gradient of the sample-specific objective function is an excellent approximation of the true gradient.
- We can show that when the learning rate decreases at a suitable rate and some mild assumptions can be made, stochastic gradient descent almost surely converges to a local minimum.

Exampl
e

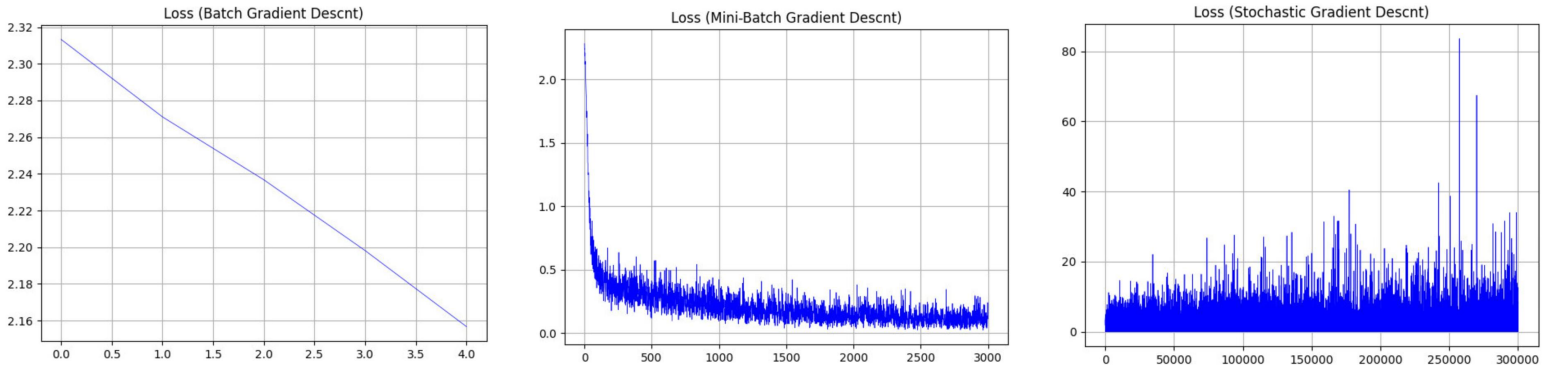


Figure:

left: Batch Gradient Descent. Entire data is used for every update (thus, 5 epochs results in 5 updates). **right**: Stochastic Gradient Descent. Every update is done based on single sample only. **centre**: Minibatch Gradient Descent. Every update is done using a batch of 100 samples.

Now we turn our attention to the Learning Rate

Learning rate Algorithm 1 : Decay

- How are we to decide the value of the learning rate?
- What happens if we choose a large value for the learning rate and let it be constant? In this case, the algorithm might come close to the optimal answer in the very first iteration but it will then oscillate around the optimal point.
- What happens if we choose a small value for the learning rate and let it be constant? In this case, it will take a very long time for the algorithm to converge to the optimal point.

Learning rate Algorithm 1 : Decay

- Choose a variable learning-rate - large initially but decaying with time.
- This will enable the algorithm to make large strides towards the optimal point and then slowly converge.
- With a learning-rate dependent on time, the update step becomes $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha_t \nabla L$.

Learning rate Algorithm 1 : Decay

- The two most common decay functions are exponential decay and inverse decay, expressed mathematically as follows:
- exponential decay: $\alpha_t = \alpha_0 e^{-kt}$
- Inverse Decay: $\alpha_t = \frac{\alpha_0}{1+kt}$
- In both of the above functions *k* controls the rate of decay.
- Another kind of decay function is the step decay where we reduce the learning rate by a constant factor every few steps of gradient descent.

Learning rate Algorithm 2 : Line search

- Line search uses the optimum step-size directly in order to provide the best improvement.
- It is rarely used in vanilla gradient descent because of its computational expense, but is helpful in some specialized variations of gradient descent.
- Let $L(\theta)$ be the function being optimized, and let $d_t = -\nabla L(\boldsymbol{\theta}_t)$.
- The update step is $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t d_t$.
- In line search the learning rate α_t is chosen at the t^{th} step so as to minimize the value of the objective function at $\boldsymbol{\theta}_{t+1}$.
- Therefore the step-size α_t is computed as $\alpha_t = \min_{\alpha} L(\boldsymbol{\theta}_t + \alpha d_t)$.

Line Search example

Iteration *i* of gradient descent $L(x_1, x_2) = {}_1x^2 + 3x^2$

1. At iteration *i* , $\nabla L(x_1, x_2) = [2x_1, 6x_2]$
2. $H(\alpha) = L(x - \alpha((\nabla L)(x))^T) = (1 - 2\alpha)^2 x_1^2 + 3(1 - 6\alpha)^2 x^2$.
3. $H^{|}(\alpha) = -4(1 - 2\alpha)x_1^2 - 36(1 - 6\alpha)x_2^2 = 0$
4. Step Size : $\alpha = \frac{x^2 + 9x^2}{2x_1^2 + 54x^2}$.

2

Line search Algorithms

- How do we perform the optimization $\min_{\alpha} L(\boldsymbol{\theta}_t + \alpha \mathrm{d}_t)$?
- An important property that we exploit of typical line-search settings is that the objective function is a unimodal function of α.
- This is especially true if we do not use the original objective function but quadratic or convex approximations of it.
- The first step in optimization is to identify a range $[0, \alpha_{\text{max}}]$ in which to perform the search for the optimum α.
- We can keep evaluate the objective function values at geometrically increasing values of α. It is then possible to narrow the search interval by using binary-search or golden-section search method

Line search Algorithms -Binary search

- Initialize the search interval $[a, b] = [0, \alpha_{max}]$.
- Evaluate the objective function at $\frac{a+b}{2}$ and $\frac{a+b+\epsilon}{2}$
- Find out whether the function is increasing or decreasing at $\frac{a+b}{2}$. Here ϵ is a small value such as 10^{-6}.
- If the objective function is found to be increasing at $\frac{a+b}{2}$, we narrow the interval to $[a, \frac{a+b+\epsilon}{2}]$ and continue the search.
- Otherwise we narrow the interval to $[\frac{a+b}{2}, b]$ and continue the search

.

Line search Algorithms-Golden-section search

- Initialize the search interval to $[a, b] = [0, \alpha_{max}]$.
- we use the fact that for any mid-samples m_1, m_2 in the region $[a, b]$ where $a < m_1 < m_2 < b$, at least one of the intervals $[a, m_1]$ or $[m_2, b]$ can be dropped. Sometimes we can go so far as to drop $[a, m_2]$ and $[m_1, b]$.
- When $\alpha = a$ yields the minimum for the objective function, i.e
  $H(\alpha)$, we can drop the interval $(m_1, b]$.
- Similarly when $\alpha = b$ yields the minimum for $H(\alpha)$ we can drop the interval $[a, m_2)$. When $\alpha = m_1$ is the value at which the minimum is achieved we can drop $(m_2, b]$.
- When $\alpha = m_2$ is the value at which the minimum is achieved we can drop $[a, m_1)$.

Line search Algorithms-Golden-section search

- The new bounds on the search interval [a, b] are reset based on the exclusions mentioned in the previous slide. At the end of the process we are left with an interval containing 0 or 1 evaluated point.
- If we have an interval containing no evaluated point, we select a random point $\alpha = p$ in the reset interval [a, b], and then another point q in the larger of the intervals [a, p] and [p, b].
- On the other hand if we are left with an interval [a, b] containing a single evaluated point $\alpha = p$, then we select $\alpha = q$ in the larger of the intervals [a, p] and [p, b].
- This yields another four points on which to continue the golden-section search. We continue until we achieve the desired accuracy.

When do we use line search?

- The line-search method can be shown to converge to a local optimum, but it is computationally expensive. For this reason, it is rarely used in vanilla gradient descent.
- Some methods like Newton's method, however, require exact line search.
- One advantage of using exact line search is that fewer steps are needed to achieve convergence to a local optimum. This might more than compensate for the computational expense of individual steps.